Assg 10: Card Games

CSci 515 Spring 2015

<2015-04-07 Tue>

Dates:

Due: Tuesday April 14, by Midnight

Objectives

- Get more practice defining and using structures.
- Practice using arrays of structs as basic tables.
- Learn to process arrays of structs.

Description

In this assignment you will extend your lab 10 work. If you did not complete the structure and enums defining your Card data type, or the function to initialize the deck of cards in the lab, you should finish those parts from the lab first.

You will add the following functionality to your deck of cards simulation. You will add a display function that can be used to display the cards in an array of cards. You will also add a shuffle function, to shuffle up the deck of cards. Finally you will add a function that will play a simple game with the deck of cards.

Perform the following tasks:

1. Create a function called `displayDeckOfCards`. This function should take an array of your Cards, and a begin and end index. The function should display the cards in the indicated index range to standard output. The begin index is inclusive, but the end index is not inclusive, so if asked to display cards from begin index 0 to end index 5, it will

display the cards at indexes 0, 1, 2, 3 and 4. See the example output below for how the output should be displayed.

2. Create a function called `shuffleDeckOfCards`. This functions should take an array of Card structures (of size 52). It will expect the cards to have already been created/initialized. This function should shuffle up the cards randomly, using the following algorithm. Iterate through all of the cards from index 0 up to 52. For each index, choose another index at random in the range from 0 up to 52. Then swap the Card at the current index of the cards array with the Card at the randomly generated index. For example, the first time through the loop the current index will be 0. Lets say you choose random index 15 to swap with. Then you will swap Card 0 with Card 15 (make sure you use a temporary variable to perform the swap). The result of doing this swap with a random card for every index in the array should be a well and randomly shuffled deck of cards.

3. Create a function called `highestDrawFour`. This function will take an array of (randomly shuffled) cards as its input. The task is to simulate 4 players drawing 4 cards from the desk. Player 0 is given the card at index 0, Player 1 the card at index 1, etc. You determine the winner in this manner. The suit of the card is the most important attribute. HEARTS beats DIAMONDS beats CLUBS beats SPADES. For example, if none of the 4 players has a HEARTS, but only 1 player has a DIAMONDS, then the player with DIAMONDS wins. If 2 or more players have equivalent best suits (for example, two players had DIAMONDS in the previous example), then you need to fall back on the face of the cards. In this case an ACE beats KING beats QUEEN beats JACK etc all the way down to a DEUCE (ACE is highest card, followed by the face cards, then the number cards by rank). Your function should display a message to standard output about which player of the 4 (Player 0, Player 1, Player 2 or Player 3) is the winner.

4. In your main function, create a deck of cards and initialize it. Then use your shuffle function to shuffle the deck of cards, and use your display function to display the 4 top cards from the shuffled deck. Finally call your `highestDrawFour` function to play a game with the top 4 cards, which should cause the winning player of the game to be displayed.

Here is an example output from running a correct implementation of this assignment.

```
$ ./assg10
000: Queen of Hearts
001: Deuce of Diamonds
002:  Five of Hearts
003:   Ten of Hearts
highestDrawFour(): winning player: 0

$ ./assg10
000:  Four of Spades
001: Three of Spades
002:  King of Diamonds
003:   Ace of Hearts
highestDrawFour(): winning player: 3

$ ./assg10
000:  Five of Spades
001:  Jack of Spades
002: Three of Diamonds
003:  Nine of Clubs
highestDrawFour(): winning player: 2
```

HINT: Finding the highest card in the game described can be a bit tricky. Here is my suggestion on how to do it. Create a function that simply compares two cards, passed in as a parameter to the function. The function should return a `bool` of True if the first card wins over the second card according to the rules, and a False if the second card is the winner. With this function you can first compare Player 0 and Player 1 card to see which one wins, the compare the winner of this came to Player 2, etc.

NOTE: Now that our programs have more functions than just the `main()` function, the use of the function headers becomes meaningful and required. Make sure that all of your functions have function headers preceding them that document the purpose of the functions, and the input parameters and return value of the function.

Assignment Submission

An eCollege dropbox has been created for this assignment. You should upload your version of the assignment to the eCollege dropbox named `Assg 10 Card Games` created for this submission. Work submitted by the due date will be considered for evaluation.

Requirements and Grading Rubrics

Program Execution, Output and Functional Requirements

1. Your program must compile, run and produce some sort of output to be graded. 0 if not satisfied.
2. 20+ pts. For the correct implementation of the function to display the cards as described.
3. 30+ pts. For correctly implementing the deck shuffling function.
4. 40+ pts. For correctly implementing the draw four game as described.
5. 10+ pts. For displaying the output of the game as described.

Program Style

Your programs must conform to the style and formatting guidelines given for this course. The following is a list of the guidelines that are required for the assignment to be submitted this week.

1. The file header for the file with your name and program information and the function header for your main function must be present, and filled out correctly.
2. A function header must be present for all functions you define. You must document the purpose, input parameters and return values of all functions. Your function headers must be formatted exactly as shown in the style guidelines for the class.
3. You must indent your code correctly and have no embedded tabs in your source code. (Don't forget about the Visual Studio Format Selection command).
4. You must not have any statements that are hacks in order to keep your terminal from closing when your program exits (e.g. no calls to system()).
5. You must have a single space before and after each binary operator.
6. You must have a single blank line after the end of your declaration of variables at the top of a function, before the first code statement.

7. You must have a single blank space after , and ; operators used as a separator in lists of variables, parameters or other control structures.

8. You must have opening { and closing } for control statement blocks on their own line, indented correctly for the level of the control statement block.

9. All control statement blocks (if, for, while, etc.) must have { } enclosing them, even when they are not strictly necessary (when there is only 1 statement in the block).

10. You should attempt to use meaningful variable and function names in your program, for program clarity. Of course, when required, you must name functions, parameters and variables as specified in the assignments. Variable and function names must conform to correct `camelCaseNameingConvention` .

Failure to conform to any of these formatting and programming practice guidelines for this assignment will result in at least 1/3 of the points (33) for the assignment being removed for each guideline that is not followed (up to 3 before getting a 0 for the assignment). Failure to follow other class/textbook programming guidelines may result in a loss of points, especially for those programming practices given in our Deitel textbook that have been in our required reading so far.