5: Shuffle a Deck of Cards

CSCI 2212 Fall 2015

1 Goals

- To use an array of class objects.
- To build upon your own code.
- To learn about serialization of a data set.

An ordinary deck of cards, such as you created in program 4, has 52 cards. In this program, you will create a deck of cards, shuffle them into a random order, then output them two ways: formatted for humans and formatted for use by another program.

2 The Data Structure

Start with a working and well-tested Card program (P4).

- Keep the CardT and SuitT type definitions from program 4, and the corresponding arrays of suit names, card names, and point values.
- Add a class definition for `DeckT`, to represent a deck of cards. Your class should have three members: an array of 52 CardT, and an int for the number of cards in the deck, and an ostream`&` for the open output file.

3 The Main Program

Do the following things in your main program:

- Call banner() and, later, bye().
- Open a stream for the output file.
- Declare a DeckT variable called `deck`. Send the name of your open stream as a parameter to the Deck constructor. When the constructor finishes, the deck should have 52 cards in it.
- Call `srand((unsigned) time(NULL))` to initialize the random-number generator, then call `shuffle()` to randomize the order of the cards.
- Print an appropriate heading, then print the deck to cout, one card per line. Format the cards in neat columns. My output looks like this:

```
Deck is shuffled.
A  Spades    points: 1
Q  Clubs     points: 10
6  Clubs     points: 6
...
3  Diamonds  points: 3
-----------------------
```

- Call `deck.serialize()` to output the cards to a file. Instead of printing the suit names, print the underlying enum constants. This makes it easier to read in the file and reuse it in an actual game.
- Properly close the output file before terminating.

4 Functions to implement.

- `CardT::CardT( SuitT suit, int rank )`
  This should be the same as in program 4.
- `ostream& CardT::print( ostream& out )`
  Modify your function from program 4 to print cards in the fixed-width format shown above/
- `ostream& CardT::serialize( ostream& out )`
  Write a simplified print function that outputs the class members directly and does not use the suits array.
- `DeckT::DeckT( ostream& deckOut )`
  Move the double-loop that created the cards from P4's main to the constructor for this class. Use the ostream`&` parameter to initialize the ostream`&` member of this class. You will need to do this in a ctor.

  In this function, there are two difficulties with C++ syntax/semantics. One involves the need for and syntax for a ctor. The other is a question of how to increment an enum type. We will discuss both of these topics on Friday.
- `ostream& DeckT::print( ostream& out )`
  Loop through the cards in the array. Print each one to the screen by delegating the job to `CardT::operator <<`.
- `ostream& DeckT::serialize( ostream& out )`
  Loop through the cards in the array. Print each one to the screen by delegating the job to `CardT::serialize()`.
- `void DeckT::shuffle()`
  Implement this shuffle algorithm:
  - Before beginning to shuffle, initialize `N` to the number of cards in the deck.
  - Execute this loop while `N>0`:
    * Get a random integer `R`, between `0` and `N-1`, by calling `rand() % N`.
    * Decrement `N`.
    * Swap the card at subscript `R` with the last card in the deck, at subscript `N`. It is possible for `R` and `N` to be equal; perform the swap anyway. In the long run, it is probably faster to do the unnecessary swap than to test for equality every time.

Testing and submission. If you understand the directions and follow them carefully, this is not a difficult program. Test your program twice:

- Capture the screen output and paste it at the bottom of your main function.
- Turn in your code and a copy of the output file. You will have at least one .hpp file (for card) and two .cpp files (for main and card). If you choose to put card and deck into different files, you will have a second .hpp file and a third .cpp file.
- Zip the file before you send it. Please note: if the UNH mail system "sees" an executable file in your messages, it will not send it and it will not notify you. So don't send executable files – that is, don't send your entire project.