Playing Card Deck

For this exercise, you will create a deck of Playing Cards, using the Playing Card class. This deck will have the normal behaviors of a deck of cards: you can shuffle it, deal from it, display the remaining cards, or reset it to a new deck. It will start with A-K of each of the four suites.

This is another example of aggregation or composition, where the new class contains instances of the previous class. In this case, the PlayingCardDeck class will contain an array of pointers to PlayingCards.

Project Design

You will need to create a new project and add the existing PlayingCard.h and PlayingCard.cpp files to it. You then will create a new class as part of this project and name it PlayingCardDeck. This will add two more files to the project, PlayingCardDeck.h and PlayingCardDeck.cpp. PlayingCardDeck.h will need to include PlayingCard.h so that you can use PlayingCard objects in your deck.

There is a test program provided in Moodle that you will use to test that your class meets the various requirements. You will need to replace the default main (hello world) with that program in your project.

Class Requirements

You will need to have an array of pointers to PlayingCards in your PlayingCardDeck. You can either define it using a constant as an array in your .h file or you can create it dynamically in your constructor. Look at the hints below for more information on these two approaches.

When the deck is created in the constructor or in reset, the array should contain cards in this order:

A-K of D, A-K of C, A-K of H, A-K of S

The deck may be shuffled before any cards are dealt. After any cards are dealt, trying to shuffle the deck will result in a warning.

When a card is dealt from the deck, a pointer to a PlayingCard object is returned from the top of the deck. The array element representing that card will be set to nullptr and the count of cards dealt will be incremented. If the deck runs out of cards, requesting a card will result in the return of nullptr.

Required Public Methods

`PlayingCardDeck();`

Default constructor. Creates an un-shuffled deck of 52 playing cards. These playing cards are to be implemented as pointers to PlayingCards. Remember, since this uses pointers, the PlayingCard objects are dynamically allocated and must be managed as such. You will want to initialize your count of how many cards have been dealt in the constructor.

The un-shuffled deck should be in the order of:

```
AD 2D 3D 4D 5D 6D 7D 8D 9D TD JD QD KD
AC 2C 3C 4C 5C 6C 7C 8C 9C TC JC QC KC
AH 2H 3H 4H 5H 6H 7H 8H 9H TH JH QH KH
AS 2S 3S 4S 5S 6S 7S 8S 9S TS JS QS KS
```

Playing Card Deck

`PlayingCardDeck(int numShuffles);`

Overloaded constructor. In addition to what is described for the default constructor, this constructor shuffles the deck numShuffles times.

One suggestion would be to create a private method named something like createDeck that can be called by both constructors and by the reset method described below. This reduces the need for duplicate code.

`~PlayingCardDeck();`

Destructor. This is necessary because of the dynamic memory allocation. When it is called, any cards remaining in the deck must be deleted. Make sure you do not attempt to delete cards that have been dealt already. If you dynamically created your array, the array should be deleted also.

As with the constructor above, creating a private method named something like deleteDeck that can be called by the destructor and reset reduces the need for any duplicate code.

`PlayingCard* dealCard();`

Returns a pointer to a PlayingCard object taken from next position of the deck. If there are no more cards in the deck, it returns a nullptr. This method should also update your count of how many cards have been dealt and should replace the card in the deck array with a nullptr.

`bool shuffle(int numShuffles);`

There is a description of shuffling below in the hint section. You need to use one of these shuffles. This method shuffles the deck **numShuffles** times. Note that, for example with the exchange shuffle, each shuffle would go through the entire deck performing random exhanges.

Shuffle should only shuffle if no cards have been dealt from the deck. If any cards have already been dealt, then the function should return false and no shuffling should occur.

`void reset();`

Resets the deck to a full deck of 52 un-shuffled cards. Remember that when you do this, you must manage the dynamically allocated cards in the deck. If you created the above described private methods for deleting and creating a deck then you can simply call those two methods. Make sure that any counts also get reset in the process.

`string getAllCardCodes();`

Returns a string containing all the value and suit codes for the deck. The cards should be displayed with a single space between each card and a line break after the 13th, 26th, 39th, and 52nd cards from the original deck. Any cards that have been dealt should not be displayed.

As an example, if 5 cards have been dealt from an un-shuffled deck, when the string is displayed, it will show the following:

```
6D 7D 8D 9D TD JD QD KD
AC 2C 3C 4C 5C 6C 7C 8C 9C TC JC QC KC
AH 2H 3H 4H 5H 6H 7H 8H 9H TH JH QH KH
AS 2S 3S 4S 5S 6S 7S 8S 9S TS JS QS KS
```

Playing Card Deck

```
int getCountUsed();
```

Returns the number of cards already dealt from the deck.

```
int getCountRemain();
```

Returns the number of cards remaining in the deck.

Program Hints

Private methods:

As described above, if you create two private methods in your class, for creating a new deck and deleting an old deck, you can call them in various places to simplify the overall design of your program and avoid duplicate code.

Displaying output:

As we discussed last term, the easiest way to return a string from a function is to use a **stringstream** variable. This requires including the **<sstream>** header and is explained here:

http://www.cplusplus.com/reference/sstream/stringstream/stringstream/

If you are not familiar with this, you can simply create and test your program using **cout** to get the output properly formatted. Once all the output is correct, you define a variable of type **stringstream** and replace the call to **cout** with the name of the variable. You then return the string from that variable.

Look at the document Creating and Shuffling a deck of cards for help with that part of the project.