ITGM 315 – Exercise 5 – Card Game Deck Only

DATE DUE: start of class 13
Data Structure Review – due no later than class 12.
Hand in through the Dropbox
Make sure your files are working with Visual Studio 2010!

Card Game – Two Player Spades

Goal – Introduce structs

You will be implementing the rules for the two-player version of the popular card game *Spades* (a partnership four-player version is available under **Start > All Programs > Games > Internet Spades** on most versions of *Windows*). This will actually be a single-player game, with the player competing against a computer (AI) opponent. If you want to try out this game with a physical deck of cards, here are the rules:

Setup: At the beginning of a hand, shuffle the deck but do not deal any cards.

Drawing the Hand: Starting with the dealer, each player first draws one card and looks at it without letting the opponent see. The player may either keep that card or discard it face-down. The player must draw a second card and do the opposite: if the player discarded the first card, they must keep the second and vic versa. The other player does the same. This process repeats until the deck is gone. At this point, each player should have 13 cards in their hand, and each player has seen 13 additional cards that the player knows are not in the opponent's hand.

Bidding: Staring with the dealer, each player must bid the number of "tricks" he or she plans to take in the hand (from 0 to 13). Bids are made out loud and recorded on a score sheet.

Play of the hand: After both players have bid, the dealer takes one card from hand and plays it face-up on the table. The other player must do the same, playing a card from the same suit if possible. (If no cards are of the same suit, they may play anything). These two cards together are called a "trick" and are taken by whoever played the highest card in the suit that was originally led. Exception: if either card is of the Spades suit, then the highest Spade wins, regardless of suit led. (Example: If the ace of diamonds is led, which is the highest card in that suit, and the other player has no diamonds but plays the 2 of spades, the spade wins the trick even though 2 is lower than Ace.) Whoever wins one trick gets to lead a card for the next trick.

Scoring: At the end of the hand, each player counts up how many tricks he or she took, and gives themselves a score as follows:

- If a player took at *least* as many tricks as the bid, they score ten points times the bid.
- If a player took extra tricks over the bid, they get one point for each overtrick.
- If a player *did not* make the bid, the player *loses* 10 times their bid
- If a player bids *zero* tricks, scoring works slightly differently. The player gets 100 points if the player takes exactly zero tricks, and the player loses 100 points if the player takes one or more tricks.

End of the Game: After scoring a hand, players reshuffle the deck and play another hand. This continues until one player reaches 500 points or drops down below -200 points. A player exceeding 500 points wins; a player going below -200 points loses. If both players cross the same threshold on the same hand, the higher score wins. If both players are tied, continue playing until either player is ahead.

This is a simple game to play in person, but it requires a bit of programming, so you are only required to implement part of the game as specified.

Remember to make small changes at a time and test your work often! It is much easier to identify problems that way.

Creating the Deck

The deck used by this game has 52 cards, each a unique combination of rank (2 to 10, jack, queen, king, and ace) and suit (clubs, diamonds, hearts, spades). Since each individual card has both a rank and a suit, a *struct* is an excellent data structure for defining an individual card.

For simplicity, store the rank as an integer from 2 to 14 (we will deal with displaying the face cards as something other than number later on). Create an enum **or** an int to store the suit. The struct should contain both of these.

The deck itself is not just a single card, but 52 of them. We will also find it useful to have an index that points to the top card of the deck (starting at the first one and incrementing as cards are drawn from the deck) so that we know where we are in the deck and when we run out of cards. These two pieces of information will be the deck. Create another struct to represent the deck. One element of this deck struct should be a one-dimensional array of the card struct type, with 52 elements. The other element should be an int, representing the current index in the array of the next card to be drawn.

Lastly, each player will have his or her own hand of cards, which is itself a card array with 13 elements. Since a player will have fewer cards in the hand as they are drawing (and after the start of play), keeping track of the number of cards in the hand is also important. Create a third *struct* to represent a single player's hand; one element of this struct should be a one-dimensional array of the card struct type, with 13 elements. The other should be an int, representing the total number of cards in hand (this will range from 0 to 13 as cards are drawn and played).

Checklist

- create an enum for card suits [enum use is optional – you may use int]
- create a struct for a single card.
- create a struct for the deck of cards.
- create a struct for the player's hand.

After you have initialized the deck and have reached this point, print the deck to the console window.

Shuffling the Deck

First we must create the cards in the deck. Right now if you declare a deck variable, it will have an array of 52 uninitialized card elements, so they could be anything. We must manually set them to something, to make sure that the deck actually has one of each card.

The easiest way to do this is in a *nested* for loop. That is, create one for loop that sets a card's rank, and inside that loop create another for loop that sets a card's suit. In *pseudocode*, it might look something like this:

```
Deck.cardIndex = 51
For i = 2 to 14 ( rank )
        For j = 0 to 3  ( or whatever values are assigned to the suit enum )
                Deck.card[Deck.cardIndex].rank = i
                Deck.card[Deck.cardIndex].suit = j
                Deck.cardIndex –
```

(Remember to look at the examples in the MATERIALS folder. ITGM315-ExampleStruct.zip)

Now you should have 52 unique cards in the deck, and the next card to be drawn in the deck struct is the very top card. (You might want to set a breakpoint and check your deck struct in the debugger to make sure.)

Print the deck to the console.

There are many ways to shuffle a deck. The simplest algorithm for a computer is to first choose a random card to be at the bottom of the deck, then choose another random card **from the remaining ones** to be next to the bottom, and keep choosing in this way until the entire deck is chosen. This results in a completely random shuffle, and it runs very fast. The pseudocode looks like this:

```
(Don't forget to call srand() before anything else)
for i = 51 down to 0
    Let r = a random number  between 0 and i ( using rand() )
    Swap the values of Deck.card[i] and Deck.card[r]
```

(Remember to look at the examples in the MATERIALS folder. ITGM315-ExampleSwapValues.zip)

After that, your deck should be shuffled. **Print the deck to the console.** (Again, you could set a breakpoint and check your deck struct to make sure this is the case. Run the program several times to ensure your shuffle is not exactly the same all the time.)

Checklist

- Declare a deck variable.
- Initialize the deck so that it has one of each card, and its card index is zero.
- Initialize the pseudo-random number generator.
- Shuffle the deck so that its cards are distributed randomly.

Drawing Cards

Starting with the human player, let's draw some cards into the player hands.

First, create variables to represent the player's and computer's hand, and initialize the number of cards in each hand to zero.

Next, we want to draw the top card off the deck and ask the player whether they want to keep it or not, but to do that, we should first find an elegant way to display the card on the screen!

Create a function that accepts a card struct as a parameter, and displays the card to the screen. By convention, cards are displayed in text as two characters: a single number or letter for the rank and then a single letter for the suit.
Specifically:

Rank	Display
2-9	2-9
10	T
11	J
12	Q
13	K
14	A

Suit	Display
Clubs	C
Diamonds	D
Hearts	H
Spades	S

Examples: two of clubs = 2C, ace of hearts = AH, ten of diamonds = TD.

The function should simply display the two-character card to cout. Reminder: use of a switch statement is the easiest way to do this.

Now that we have a way to display a card, "draw" the top card and display it to the screen, and ask the player if they want to keep it. If the player says yes, add that card to the hand (this involves setting the card values in the hand and incrementing the number of cards in the hand), then draw a second card and display it so that the player knows what was discarded. If the player says no, draw the second card and add that to the hand instead.

Then the computer opponent should "draw" the next two cards. For simplicity, let's have the computer always keep the first card, so all we have to do is add the top card of the deck to the computer's hand and ignore the next card. The computer's cards should not be displayed to the player.

Put this cycle of the human and computer drawing cards in a loop continuing until the deck is exhausted. There are many ways to implement this: a while or do-while loop that stops when the card index in the deck *struct* exceeds 51 (or when both players' hands contain 13 cards), or a for loop that iterates exactly 13 times. It is up to you to decide which method is the most elegant.

When you are done, **display the player's entire hand to the screen.** Part of a sample run of the game might look like this (before this you will display the deck and then the shuffled deck):

```
You drew TC. Do you keep it (Y of N)? Y

You discarded 8D.
You drew 2C. Do you keep it (You N)? M

Enter either Y for Yes (keep it) or N for No (discard it):  N

You drew and kept 3C.
You drew AS. Do you keep it (Y or N)? Y

You discarded KS.
You drew TS. Do you keep it (Y or N)? Y

You discarded 5H.
You drew QH. Do you keep it (Y or N)? Y

You discarded AD.
You drew 9D. Do you keep it (Y or N)? N

You drew and kept 2D.
You drew 6C. Do you keep it (Y or N)? N

You drew and kept 8H.
You drew KC. Do you keep it (Y or N)? Y

You discarded TH.
You drew JH. Do you keep it (Y or N)? Y

You discarded AH.
You drew 7H. Do you keep it (Y or N)? N

You drew and kept 2S.
You drew 3S. Do you keep it (Y or N)? Y

You discarded KD.
You drew 6S. Do you keep it (Y or N)? Y

You discarded 9S
You drew 4H. Do you keep it (Y or N)? N

You drew and kept 5H.
Your hand is: TC 3C AS TS QH 2D 8H KC JH 2S 3S 6S 5H
```

Checklist

- create and initialize variables of hand *struct* type to represent player and computer hands.
- create a function to display a card to the screen.
- print the deck to the console before and after shuffling!
- give the player the choice of keeping or discarding the first card, then add either that card or the next one to the player's hand
- have the computer keep its first card and discard its second one, without notifying the player.
- enclose the previous two steps in a loop that continues until both players have kept 13 cards and discarded 13 cards each, so that the deck is empty and both players' hands are full
- when the player has 13 cards, display the entire hand on the screen.

Options: For extra credit, also create a function to sort the player's hand before it is displayed to the screen. Sort first by suit (clubs, then diamonds, then hearts, then spades, and then rank (low to high)). The hand from the earlier example would sort like this:
Your hand is 3C TC KC 2D 5H 8H JH QH 2S 3S 6S TS AS

At this point, your program should be able to create a deck of cards, shuffle it, and allow the human and computer opponents to draw their hands from it. When done, it should display the deck, the shuffled deck and the player's hand (optionally sorted). This is all that is required.

Submission: Please remember to name your zip file properly LastnameFirstnameExercise5.zip

Grading Guidelines:
Design and Debugging (20%)
Use of Structs (20%)
Use of Arrays or Vectors (20%)
Coding style (20%)
Use of Comments (15%)
Data Structure Review (5%)
Extra Credit (10%)