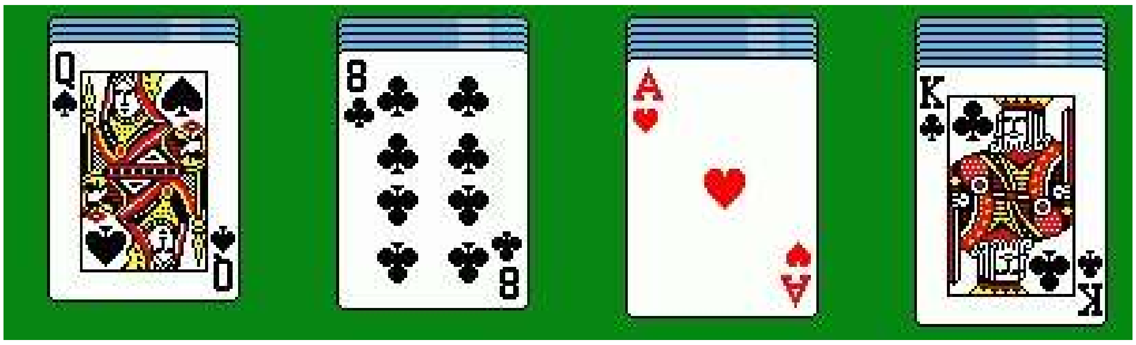
CSCE 1101 Fall 2022: Fundamentals of Computing II
Assignment #4

Dr. Amr Goneid ***Date: Thu Nov 17, Due: Thu Nov 24, 2022***

In this assignment, you will experience the design and implementation of an ADT class library with a game application.

The Problem:

A standard 52-card deck contains 4 suits: Heart (♥), Diamond (♦), Spades (♣) and Club (♠),.
In each suit, there are 13 cards numbered (ranked) as: 2,3….,10, J,Q,K,A.
A fresh deck of cards would be ordered as follows:
(2,3….,10, J,Q,K,A)$_{\text{heart}}$, (2,3….,10, J,Q,K,A)$_{\text{diamond}}$, (2,3….,10, J,Q,K,A)$_{\text{spades}}$, (2,3….,10, J,Q,K,A)$_{\text{club}}$



In a simple card game, a standard 52-card deck is dealt to 4 players such that each player has 13 cards. Each player in turn puts a card face down on the table and when all four cards are on the table, the cards are turned face up. Whoever turned the higher value card wins the round and his score is accumulated. Players then repeat the round until all 13 rounds are finished. The object of the game is to get the highest score after the 13 rounds.
The value of a card is its rank (2 or 3 or ….) according to the following table.

Rank	2	3	4	5	6	7	8	9	T(ten)	J	Q	K	A
Value	2	3	4	5	6	7	8	9	10	11	12	13	14

If the cards of two players have the same rank, the suits give extra weight such that Club > Spades > Diamond > Heart (see the following table).

Suit	Heart (♥)	Diamond (♦)	Spades (♣)	Club (♠)
Added Value	3	4	5	6

In this assignment, you are to design and implement a program to play this game according to the following steps:

- Create a fresh (ordered) deck of cards.
- Shuffle the deck.
- Cut the deck (exchange bottom half with top half).
- Deal all the cards to four players (one card to a different player at a time).
- Play 13 rounds to accumulate scores and identify the winning player.

Design Guidelines:

Design and implement a class ***Deck*** for an ADT that represents a deck of cards.
A small class ***cardType*** can be used to represent a single card. It has the public members:
{ char suit; char rank; }. This small class can be defined in the the public part of the bigger class **Deck.**
The suit (Heart , Diamond , Spades , and Club) can be represented by the characters with ASCII values of 3,4,5 and 6, respectively. The card rank is set according to the table shown before.
An array of ***cardType D[ ]*** of size 52 can be used to represent the deck. The top is initially set to -1. It is incremented when a card is added and decremented when the card is removed.

The following are the basic (public) methods for the ***Deck*** class:

- **Deck ( )** (constructor, to set top to -1)
- **~Deck( )** (destructor)
- **Create_Fresh_Deck ( )** (creates a fresh "ordered" deck)
- **EmptyDeck( ) → bool** (true if a deck is empty)
- **RemoveCard( ) →** (remove and return a card from the top of the deck)
- **AddCard(card)** (adds a card to the top of the deck)
- **Shuffle_Deck ( )** (shuffles the deck)
- **Cut_Deck ( )** (exchange bottom half with top half).
- **Display_Deck( )** (to display the cards in the deck)

The small class ***cardType*** with the two public members **{ char suit; char rank; }** is also defined in the public part of the class ***Deck.***
Class ***Deck*** will have the following private members:

```
cardType D[52];          // Array containing the Deck of cards
int top;                 // Top of Deck
void swap(int i, int k); // prototype for a utility function to swap cards (i) & (k)
```

Implementation of the class *Deck*

The following is a sketch of the implementation file for the class ***Deck.*** You have to complete the missing implementations.

```
Deck::Deck()          // Constructor
{ set top to -1 }
```

```
Deck::~Deck()                   // Destructor
{ }
```

```
bool Deck::EmptyDeck() const            // return True if Deck is empty
{ return true if top is -1 else return false }
```

```
void Deck::AddCard(const cardType &c) // Add a card (c) to top of deck
{ increment top then store card c in D[top] }
```

```
cardType Deck::RemoveCard()              // Remove and return top card
{ Get top card , decrement top then return the card }
```

```
void Deck::Create_Fresh_Deck()           // Create a fresh Deck
{
        cardType c;
        string v = "23456789TJQKA";
        top = -1;
        for(int i = 0; i < 4; i++)
                for(int j = 0; j < 13; j++) {
                   c.suit = char(i+3);   // Card Suit: ASCII values of 3,4,5 or 6
                   c.rank = v[j];        // Card rank
                   AddCard(c); }         // Add card to deck
}
```

```
void Deck::swap(int i, int k)         // Swap cards (i) and (k)
{ Swap D[i] with D[k] }
```

```
// Shuffle Deck.
// For i from 51 down to 1 generate a random integer k whose value is from 0 to i.
// Then swap cards (i) and (k).
void Deck::Shuffle_Deck()
{
        srand ( (unsigned) time (NULL) );  //Initialize Random Number Generator
        for(int i = 51; i >= 1; i--) {
                int k = rand( )%(i+1);  // Random integer from 0 to i
                swap(i,k); }
}
```

```
void Deck::Cut_Deck()        // Cut Deck: Exchange upper half with lower half
{ Swap cards in locations 0 to 25 with those in locations 26 to 51 }
```

```
void Deck::Display_Deck() const          // Display Deck
{ If Deck is not empty, display suit and rank of each card in the Deck }
```

The Application Program (The Game)

The following gives the main steps for the game program. You have to complete the missing implementations to run the game.

```
int main()      // Application  (client) program starts here
{
  // Before starting the game, you need to set up the following:
  // Declare the Number of Players NP (4 players)
  // A string v of card ranks = "23456789TJQKA"
  // The card deck cdeck of type Deck
  // Hand[NP]: Hands of each player, an array of Decks, one for each player
  // A card c of type cardType
  // cards[NP]:  Array of type cardtype for cards put on the table by each player
  // score[NP]  An int array for the score of each player
  // int N : used for next player
  // int p: player number
  // int wp: winning player in a given round
  // int win:  the amount of win in that round

  // Prepare Deck
            1.  Create a fresh deck (cdeck)
            2.  Shuffle the whole deck
            3.  Cut the deck
            4.  Display the deck (Optional)

  // Distribute all cards to NP players
        N = 0;
        while(!cdeck.EmptyDeck())          // While there are still cards left
        {
            p = N % NP;                    // select player (0,1,2,3,0,1,2,3, etc)
            pick a card from top of deck
            Distribute card to hand of player (p)
            N++;                           // next player
        }

  // Set scores of all players to zero.
  // Display the hands (Optional)
        ......................................
        ......................................
  // Game starts here ________________________________________
        while(!Hand[0].EmptyDeck())            // while first player still has a card
        {
            // Start a round. Each player puts a card face down on table
            for(p = 0; p < NP; p++)      cards[p] = Hand[p].RemoveCard();
            // Cards are turned and winner player is defined (wp)
            .......................................
            .......................................

            // display cards on table and winner card
            .......................................
            .......................................

            // Accumulate score for winning player
            .......................................
            .......................................


            // Stop to view results for this round, then continue to next round.
            .......................................


        }
        // Game ends here ________________________________________
        // Display Final scores
        .......................................
        .......................................
        return 0;
}
// End of application program
________________________________________________________
```

What you should deliver:

- Two library files for class ***Deck***, a header file (.h) and an implementation file (.cpp)
- The application file (.cpp)
- The executable project file (.exe)
- Output for a sample game.