CSCI 2170 OLA 3
(Part 1 soft copy due midnight, Sunday, February 20th;
hard copy due beginning of class, Monday, February 21st.

Part 2 soft copy due midnight Tuesday, March 1st; hard copy due beginning of class, Wednesday, March 2nd)

Write a C++ program which will play a card game called "Hearts". Hearts is a trick taking game in which the objective is to avoid winning tricks containing hearts; the queen of spades is even more to be avoided.

The program plays "Hearts" with four players: computer 1, computer 2, computer 3, and the user. The winner of the game is the person who has the lowest number of points after the game is over. Points in the game are calculated as follows: each card of suit HEART has points: those with face value less than 10 is worth 5 points, and the rest of the HEART cards worth 10 points. In addition, the Queen of Spades worth 100 points, the Club of Jack worth -100 points.

Each player is dealt 13 cards at the beginning of the game. The person who holds the 2 of CLUB must lead it to the first trick. The suit on the first card played on each hand is called the leading suit for that hand. Then, the remaining three players must each play a card of the same suit if they have one. If not, they can play a card from any other suit. For each hand, the player with the highest valued card of the leading suit must collect the points. Then, this player must begin the next round in a similar fashion with a new leading suit. Therefore, 13 different "hands" will be played. The game is over after all 13 hands have been played.

Your program simulates this game by showing the user's cards at the beginning of each round (in a nice form such that the user can view all his or her remaining cards easily). At each round, your program must display the cards played by each player, and show points accumulated for each player, and indicate who will lead the next hand.

The strategy for card selection can be different between the user and the three computers. The user may select any card to start a new round of game when it is his or her turn, or select an appropriate card of the required suit from his remaining cards. For the three computers, when it is their turn to start the hand, they will always select the card at the top of the remaining card list. When they are not leading a hand, they always select the first available card that meets the suit requirement.

Your program must create a class called "CardClass". The data member of this class will be the deck of cards implemented in terms of an array of structs of size 52. The class should at least include the following member functions:

1. Default constructor -- which creates the deck of cards by assigning appropriate suit, value, and points for each card. A deck of cards contains four suits: Hearts, Diamonds, Spades, and Clubs. The cards are listed in order of their rankings: Ace, King, Queen, Jack, 10, 9, 8, 7, 6, 5, 4, 3, 2. This corresponds to the **FormCards** function in OLA2.
2. ShuffleCard. This corresponds to the **ShuffleCards** function in OLA2.
3. DealCard. This method deals out (returns) **one card only,** it should also decrement the number of cards remaining in the deck by one.

Part 1 (100 points)
Implement a CardClass, including the header file and the implementation file for the class. **Make sure to include detailed description (description, pre-condition, post-condition) of each method in the class in the header file.**
Your *CardClass* should include:

(i) number of cards remaining in deck. (This variable keeps track of how many cards remain in the deck).
(ii) the deck of cards, i.e., an array of card struct type.

Your CardClass methods should include:

(i) default constructor that creates the deck of cards – assign values to each card in the deck;
(ii) shuffle the cards,
(iii) deal card.

Write a client program, heartsA.cc, that :

(i) creates an object of CardClass;
(ii) deals cards to 4 players (2-D array of cardType, same as in OLA 2)
(iii) sort each player's cards
(iv) displays the cards for each of the 4 players.

This program should be very similar to the main function in the program you have in OLA2.

Instructions to submit your program

Soft copy:

- login the ranger system with www.cs.mtsu.edu/nx,
- login to PeerSpace through the web browser provided by the ranger system, click on *tools*|*Assignments* to submit your softcopy.

Hard copy:
To turn in part A, create a script file including:

- Create a script file by following the steps below: First, navigate to the directory where your program source file is located, then follow the steps below:

```
ranger% script log3A
ranger% pr –n –t –e4 CardClass.h
ranger% pr –n –t –e4 CardClass.cpp
ranger% pr –n –t –e4 heartsA.cc
ranger% aCC CardClass.cpp heartsA.cc –o RunA
ranger% RunA
ranger% exit

ranger%lph CardClass.h      ← print the source files
ranger%lph CardClass.cpp
ranger%lph heartsA.cc
```

- Enclose the hardcopy of the program and the program evaluation sheet in a folder

Part 2 (100 pts)
Add the second class "PlayerClass". The header file for the class is given below.
You may copy it into your account with:

cp ~cen/code/playerclass.h playerclass.h

In addition, you may download the implementation of two of the member functions here:

cp ~cen/code/playerclass.cpp playerclass.cpp

Complete the client program by adding code to play the entire "hearts" game. In your client program, define an array of PlayerClass objects as:

playerclass player[4];

player[0] is the user, player[1], player[2], and player[3] are the three computer players.

If your program uses a different strategy of selecting card to play for the computer players, describe the strategy in the program description section.

```
class PlayerClass
{
   public:
     // default constructor
     // post-condition: count is assigned 0, score is assigned 0
     PlayerClass();

     // add one card to the player's hand
     // pre-condition: player has less than 13 cards, a card is supplied as input parameter
     // post-condition: player has one more card in hand, count is increased by 1
     void      AddCard(CardStruct);

     // prints out the current cards in the player's hand
     void      DisplayCards();

     // select to play the first card that has the suit supplied from the client program
     // if no card can be found that has the suit supplied by the client program, the first card
     // from the hand is played
     // pre-condition: there are >= 1 cards in hand
     // post-condition: a card is returned/played, count is decremented by 1
     CardStruct  FollowOneCard(suitType s);

     // Plays the first card of a round. The top card in the deck is selected and returned/played
     // pre-condition: there are >= 1 card(s) in the player's hand
     // post-condition: one card is played/returned, count is decremented by 1
     CardStruct  StartOneHand();

     // Checks to see if the player should lead the
     // first round in the game, e.g., check whether the player has 2 of club
     // pre-condition: the hand is full (have 13 cards)
     // post-condition: if this player has 2 of club, true is returned, otherwise, return false.
     bool              IsFirstLead();

     // The current score of the player is returned
     int       GetScore();
```

```
        // The points from the current round is added to the current player's score
        // pre-condition: the score from the current round is sent as input parameter
        // post-condition: the player's score is increased by the points sent
        //              in through the parameter
        void       AddScore(int);

        // Plays the card selected by user. After the cards in the player is displayed to the user, the
        //  user selects a card by entering the number corresponding to the card.
        // pre-condition:  the card number selected by the user is supplied
        // post-condition: the card corresponding to the user choice is played/returned.
        //                    the number of cards in player's hand is decremented by 1
        CardStruct  PlaySelectedCard(int choice);

        // Return the number of cards the player has
        // post-condition:   the number of cards is returned
        int        GetCount();

        // Checks to see if the card the user chooses is a valid choice. If the user has cards of the
        // leading suit, the card he chooses to play has to match the leading suit.
        // pre-condition:  the player's choice of card number and the current leading suit
        //                    are sent as input parameters
        // post-condition: returns true if
        //        (1) user has cards of leading suit and the choice card is of that suit
        //        (2) user does not have card of leading suit
        //       and returns false otherwise
        bool       IsValidChoice(suitType, int);

    private:
      CardStruct   hand[MAX_PLAYER_CARDS];
      int          count;   // keeps record of the number of cards the player has in hand
      int          score;   // keeps score for the player
};
```

Instruction to turn in the program:

Soft copy:

- login the ranger system with www.cs.mtsu.edu/nx,
- login to PeerSpace through the web browser provided by the ranger system, click on *tools*|*Assignments* to submit your softcopy.

Hard copy:

To turn in part B, create a script file including:

- Create a script file by following the steps below: First, navigate to the directory where your program source file is located, then follow the steps below:

```
ranger% script log3B
ranger% pr –n –t –e4 CardClass.h
ranger% pr –n –t –e4 CardClass.cpp
ranger% pr –n –t –e4 PlayerClass.h
ranger% pr –n –t –e4 PlayerClass.cpp
```

```
ranger% pr –n –t –e4 heartsB.cc
ranger% aCC  CardClass.cpp PlayerClass.cpp heartsB.cc –o Run
ranger% Run                    ← play to completion of a game
ranger% exit
ranger% lph CardClass.h      ← print out the source files
ranger% lph CardClass.cpp
ranger% lph PlayerClass.h
ranger% lph PlayerClass.cpp
ranger% lph heartsB.cc
```

- Enclose the hardcopy of the program and the program evaluation sheet in a folder