

INB371 & INN371 – Data Structure and Algorithms

Assignment 1

Poker Hand Evaluator

Due Date: 11:59 pm – 28 April, 2013

Weighting: 20%

Individual Assignment

Introduction

The game of Poker has many variants with respect to the mechanics of the play of the game. Almost all variants, however, rely on the standard ordering of poker hands which is described as follows:

- High Card — does not fit into any other ranking below. When comparing two High Card hands, the ranks of the highest cards in the two hands are first compared. If there is a tie, the second highest cards in each hand are compared, and so on. Example: QS, JH, 9C, 7H, 3D.
- One Pair — two cards of the same rank. When comparing two One Pair hands, the hand with the higher ranked pair beats the hand with the lower ranked pair. In case of a tie, the remaining three cards are used as tie-breakers, compared in descending order of their ranks (as with the High Card ranking). Example: 6D, 6H, QD, 9H, 4S.
- Two Pair — two pairs of cards of the same rank. When comparing two Two Pair hands, the higher pair is first compared, then the lower pair, and finally the rank of the remaining fifth card. Example: JH, JS, TS, TD, 8S.
- Three of a Kind — three cards of the same rank. When comparing two Three of a Kind hands, the hand with the higher ranked three-equal-rank-cards beats the other. In case of a tie, the remaining two cards are used as tie-breakers, compared in descending order (as with the High Card ranking). Example: 5S, 5H, 5D, JH, 6D.
- Straight — five cards in consecutive rank, noting that an ace can either be counted above a king or below a two, but not both (i.e., wrapping is not allowed). When comparing two Straight hands, the rank of the highest card in the straight is used to determine which hand beats the other (in the case of A, 2, 3, 4, 5, the highest card is considered to be 5). Example: QH, JC, TH, 9D, 8D.
- Flush — five cards of the same suit. When comparing two Flush hands, the rank of the highest card is first considered, then the second highest, and so on (as with the High Card ranking). Example: AS, JS, 8S, 6S, 5S.
- Full House — three cards of the same rank and two cards of the same (but different to the three equal rank cards) rank. When comparing two Full House

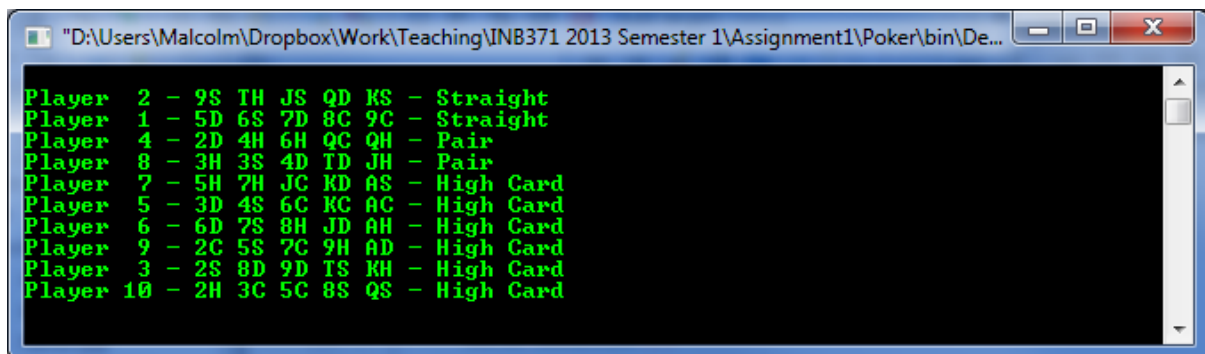
hands, the rank of the three-equal-rank-cards cards is first compared, then the rank of the pair. Example: 7S, 7H, 7C, JC, JH.

- Four of a Kind — four cards of the same rank. When comparing two Four of a Kind hands, the hand with the higher ranked four-equal-rank-cards beats the lower ranked one. In case of a tie, the rank of the fifth card is used as a tie-breaker. Example: 4C, 4D, 4H, 4S, TD.
- Straight Flush — a hand that is both a Straight and a Flush. The same tie-breaker as for a Straight hand is used to compare two Straight Flushes. Example: TH, 9H, 8H, 7H, 6H.

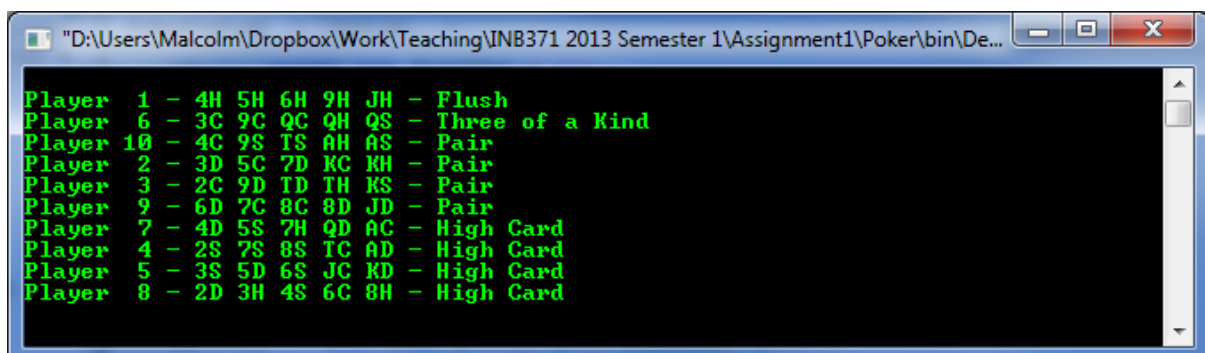
Task

Your program will do the following:

1. Create a standard deck of 52 playing cards with cards ranked 2, 3, 4, 5, 6, 7, 8, 9, T (ten), J (jack), Q (queen), K (king) and A (ace), in four suits C (clubs), D (diamonds), H (hearts) and S (spades).
2. Shuffle the deck into a random ordering
3. Deal five (5) cards to ten (10) players. Cards are dealt in player order from player 1 to player 10: each player receives one card before the first player receives their second card and so on.
4. The program then evaluates the hands to determine the type of hand the player has received, e.g. Full House, orders the hands from highest to lowest and displays them in descending order. Some examples of output follow:



```
"D:\Users\Malcolm\Dropbox\Work\Teaching\INB371 2013 Semester 1\Assignment1\Poker\bin\De...
Player 2 - 9S TH JS QD KS - Straight
Player 1 - 5D 6S 7D 8C 9C - Straight
Player 4 - 2D 4H 6H QC QH - Pair
Player 8 - 3H 3S 4D TD JH - Pair
Player 7 - 5H 7H JC KD AS - High Card
Player 5 - 3D 4S 6C KC AC - High Card
Player 6 - 6D 7S 8H JD AH - High Card
Player 9 - 2C 5S 7C 9H AD - High Card
Player 3 - 2S 8D 9D TS KH - High Card
Player 10 - 2H 3C 5C 8S QS - High Card
```



```
"D:\Users\Malcolm\Dropbox\Work\Teaching\INB371 2013 Semester 1\Assignment1\Poker\bin\De...
Player 1 - 4H 5H 6H 9H JH - Flush
Player 6 - 3C 9C QC QH QS - Three of a Kind
Player 10 - 4C 9S TS AH AS - Pair
Player 2 - 3D 5C 7D KC KH - Pair
Player 3 - 2C 9D TD TH KS - Pair
Player 9 - 6D 7C 8C 8D JD - Pair
Player 7 - 4D 5S 7H QD AC - High Card
Player 4 - 2S 7S 8S TC AD - High Card
Player 5 - 3S 5D 6S JC KD - High Card
Player 8 - 2D 3H 4S 6C 8H - High Card
```

```
"D:\Users\Malcolm\Dropbox\Work\Teaching\INB371 2013 Semester 1\Assignment1\Poker\bin\De...
Player 4 - 5C 5D JD JH JS - Full House
Player 9 - 2C 2H TC KH KS - Two Pair
Player 7 - 2S 3C 3D QC QS - Two Pair
Player 1 - 6C 7D 9D AC AH - Pair
Player 2 - 9S TS JC QD QH - Pair
Player 6 - 3H 7C 8D 8H AD - Pair
Player 8 - 4H 6D 6S 9H TH - Pair
Player 3 - 3S 4C 4S 5H KC - Pair
Player 5 - 2D 6H 7S TD AS - High Card
Player 10 - 4D 5S 7H 8S KD - High Card
```

```
"D:\Users\Malcolm\Dropbox\Work\Teaching\INB371 2013 Semester 1\Assignment1\Poker\bin\De...
Player 10 - 6D 7C 7D 7H 7S - Four of a kind
Player 8 - 9S TH JS QC KC - Straight
Player 2 - 4H QD QH QS AS - Three of a Kind
Player 1 - 4C 4D 5C 5H AD - Two Pair
Player 7 - 2C 3H TS AC AH - Pair
Player 5 - 8S 9C 9H JD KD - Pair
Player 6 - 3D 6C 6S TC JC - Pair
Player 3 - 2H 4S 8C JH KS - High Card
Player 9 - 3C 6H 8D TD KH - High Card
Player 4 - 2S 3S 5D 8H 9D - High Card
```

```
"D:\Users\Malcolm\Dropbox\Work\Teaching\INB371 2013 Semester 1\Assignment1\Poker\bin\De...
Player 6 - TH JH QH KH AH - Straight Flush
Player 1 - 3D 3H 9C 9D JS - Two Pair
Player 10 - 6S TC TS KS AC - Pair
Player 3 - 4C 5D 8H 8S 9S - Pair
Player 8 - 2C 6H 7H 7S TD - Pair
Player 4 - 3C 4D 5C 5S QS - Pair
Player 5 - 2S 4S 9H KD AS - High Card
Player 2 - 6D 8C JD QC AD - High Card
Player 9 - 2D 3S 6C 7C KC - High Card
Player 7 - 4H 5H 7D 8D JC - High Card
```

Design Decisions

The following design decisions may aid in your implementation. These guidelines do not have to be followed.

The solution that produces the above code was built by creating the following classes:

- **Random** – produces random integers
- **Card** – encapsulates a rank (2 to Ace) and suit (Diamonds, Clubs, Hearts, Spades)
- **Deck** – encapsulates a collection of 52 individual **Card** objects
- **Hand** – encapsulates a collection of 5 **Card** objects that are dealt from the **Deck** of **Card**, the player identifier (1 to 10), and information about the type of hand the cards make up
- **CardComparer** – responsible for providing an ordering for **Card** objects
- **HandComparer** – responsible for providing an ordering for **Hand** objects

The Random Class

This class is based on the exercises included in Workshops 3 and 4. To ensure that different random values are generated on successive runs, the **Randomise()** method should be called by the constructor.

The Card Class

This class is based on the exercises included in Workshops 3 and 4. The header file for this class, should also include definitions for the enumerations **Rank** and **Suit**.

| | |
|--------------------------|--|
| Card() | No argument constructor |
| Card(Rank, Suit) | Constructor that sets the rank and suit for a Card object |
| ~Card() | Destructor – does nothing as no objects are created dynamically by the constructor of this class |
| Rank GetRank() | Accessor for the Rank instance variable |
| Suit GetSuit() | Accessor for the Suit instance variable |
| string ToString() | Produces a string representation of this Card |

The Deck Class

This class is based on the exercises included in Workshops 3 and 4.

| | |
|-----------------------------|--|
| Deck() | No Argument constructor – Creates a dynamic array of Card* objects and initialises them. Initialises cardsDealt to 0. Note that the dynamic array should contain <i>pointers</i> to Card to minimise the overhead of copying during the shuffling procedure. See Lecture 5, Slides 30 and 31 for an example of how this is done. |
| ~Deck() | Destructor – deletes the contents of the vector of Card* |
| Card* DealNextCard() | Returns a pointer to the next Card object from the deck |
| void Shuffle() | Shuffles the cards in the deck. This can be achieved by selecting two random indexes in the array and swapping the references around, putting the cards out of their original order. |
| void DisplayDeck() | Outputs the Card objects in the deck to the screen. This method is useful during testing to ensure that shuffling is being achieved. |

The Hand Class

The **Hand** class is responsible for, among other things, storing the cards dealt to a particular player.

The header file for the **Hand** class should declare an enumeration type for the type of hand (the table below assumes this enumeration is named **HandType**) that has been dealt, from the lowest hand i.e. High Card up to the highest hand i.e. Straight Flush.

The **Hand** class is also responsible for determining the type of hand the five cards dealt to the hand make up. The class could encapsulate:

- A player identifier (**int**)
- A **vector** of pointer to **Card** – a **vector** can make use of the **sort** method from the algorithm library. It will rely on the **CardComparer** class to make this comparison.
- A value to store the **HandType** for this **Hand**. Some discussion of how hand types can be determined is included below.
- A value to rank the worth of this **Hand**. The value will provide an ordering for different hands with the same **HandType**, e.g. a pair of Kings is ranked higher than a pair of Queens and a hand made up of (2C 4D TH KH KC) must be ranked higher than (2D 4H 9H KD KS). A discussion of how such a value could be generated is included below.

| | |
|-------------------------------|--|
| Hand(int) | Constructor – sets the player identifier for this hand. The encapsulated vector should contain <i>pointers</i> to Card . |
| ~Hand() | Destructor – deletes each Card* in the Hand |
| void AddCard(Card*) | Adds a Card* to the Hand vector |
| void Evaluate() | Responsible for determining the type of hand the cards make up and for providing some means of ranking the hand when it is compared with other hands. The type of hand is easier to determine if the cards in the hand are in sorted order of their rank. |
| int GetValue() | Accessor for the “value” of a Hand , i.e. for comparison with other Hand objects. |
| HandType GetHandType() | Accessor for the HandType |
| string ToString() | Creates a string representation of a Hand to output its player identifier , its Card objects and a string representation of its HandType . |

Determining the HandType of a Hand

The HandType of a Hand should be checked carefully, assigning only the maximum possible HandType value to the Hand. For example, a hand that contains a Full House also contains a Three of a Kind and a Pair, but Full House is the maximum possible HandType.

Determining the value of a Hand

The value of a hand is needed when there are multiple hands in a game which have the same HandType. The values is needed to provide the proper ordering.

If we consider a Flush, the hand contains five cards of the same suit. The only way to compare two Flush hands is to check the highest card in both hands, and if they are the same, to check the second highest and so on, until one of the cards in one of the hands has a higher value. If the hands both have the same ranked cards from different suits, they would be considered equal.

If we consider a Straight, the only card that is important in determining if one Straight is higher than another, is the highest card in the Straight.

If we consider a Pair, we have to first check the value of the paired card, then the values of the other cards in the hand in descending order. For a Pair, we have to consider four values.

We can make use of this observation when combined with the knowledge that there are 13 ranks for the cards (a Two has enumeration value of 0 and an Ace has an enumeration value of 12) to combine the values into one summary value.

If we take some examples of hands that contain Pair (suits are left out):

2 2 3 4 A

2 2 3 K A

2 3 K K A

2 3 K A A

We can see that these hands are ordered from lowest to highest. If there were only 10 cards in a suit and Ace was ranked highest (9), King was ranked second highest (8), 3 and 2 were ranked second lowest (1) and lowest (0), we could combine these rankings as follows:

$$2\ 2\ 3\ 4\ A = 0 \times 10^3 + 9 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 = 921$$

$$2\ 2\ 3\ K\ A = 0 \times 10^3 + 9 \times 10^2 + 8 \times 10^1 + 1 \times 10^0 = 981$$

$$2\ 3\ K\ K\ A = 8 \times 10^3 + 9 \times 10^2 + 1 \times 10^1 + 0 \times 10^0 = 8910$$

$$2\ 3\ K\ A\ A = 9 \times 10^3 + 8 \times 10^2 + 1 \times 10^1 + 0 \times 10^0 = 9810$$

We can see that this combined value has ordered the hands correctly.

Because there are 13 cards per suit, we must use 13 instead of 10 in such a calculation, but if we use the values of the cards in the correct order, we will get a proper ranking for each hand type. This combined value cannot be used to rank hands by itself. If we had two hands, 8 T Q K A and 2 2 3 4 5, the summary value for the first hand is 368699 and the summary value for the second hand is 90246, but the second hand beats the first hand.

The CardComparer Class

The **CardComparer** class is responsible for providing an ordering for determining if one **Card** object is less than another **Card** object based on the rank of the card i.e. (2 is low and Ace is high). The details for how such a class works will be covered in Lecture 6.

| | |
|---------------------------------------|---|
| bool operator() (Card*, Card*) | This method returns true iff the rank of the first Card is less than the rank of the second Card |
|---------------------------------------|---|

The HandComparer Class

The **HandComparer** class is responsible for providing an ordering for determining if one **Hand** object is less than another **Hand** object. The details for how such a class works will be covered in Lecture 6.

| | |
|---------------------------------------|---|
| bool operator() (Hand*, Hand*) | This method returns true if and only if the first Hand has a lower rank than the second Hand |
|---------------------------------------|---|

The PokerEvaluation Program

The program should:

- Declare and initialise a **Deck**
- Declare a **vector** of pointer to **Hand**
- Create and add an empty **Hand** for each player to the **vector**
- Shuffle the **Deck**
- Deal 5 **Cards** to each player
- Evaluate each **Hand**
- Sort the **vector** of pointer to **Hand** based on the ordering provided by **HandComparer**
- Display each **Hand** in the format given in the sample runs

A partially completed file has been provided (**PokerEval.cpp**) which includes the ability to read a predetermined **Deck** into the program. Some text files will be provided so that you can check the reliability of your program. For example, the file **three.txt** will produce the following output when run from the command line as **PokerEval three.txt**. The program will terminate with an error if the file given as a command line argument does not exist.

```
Administrator: C:\Windows\system32\cmd.exe
D:\Temp>Poker three.txt
Player 2 - 7H 9D 9H 9S TS - Three of a Kind
Player 8 - 3D 5S 6S AD AH - Pair
Player 4 - 2D 3H 3S 5H JD - Pair
Player 5 - 2C 2H 3C 7D TD - Pair
Player 7 - 4S JC QC KS AS - High Card
Player 6 - 4C 8H QS KD AC - High Card
Player 3 - 4D 6C JH QH KC - High Card
Player 1 - 5D 7S 8D QD KH - High Card
Player 10 - 4H 5C 6H TH JS - High Card
Player 9 - 2S 7C 8C 9C TC - High Card

D:\Temp>Poker four.txt
Error: Could not find file

D:\Temp>
```