[OOP]Assign4

Student ID: 20152008

Name: Kusdavletov Ernar

Date: 2017.04.18

1. Descriptions

A poker deck contains 52 cards. Each card has a suit of either clubs, diamonds, hearts, or spades (denoted C, D, H, S in the input data). Each card also has a value of either 2 through 10, jack, queen, king, or ace (denoted 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A). For scoring purposes card values are ordered as above, with 2 having the lowest and ace the highest value. The suit has no impact on value. A poker hand consists of five cards dealt from the deck. Poker hands are ranked by the following partial order from lowest to highest.
The program reads the input 5 cards for each hand as strings. Then, convert them into numbers and determine the type of cards combination. After, it checks which hand has better combination, and if they are same compares by highest cards.

2. Code

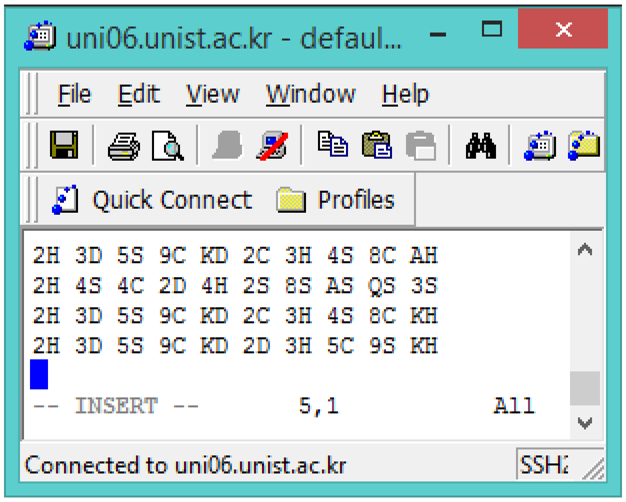
```
#include <iostream>
using namespace std;

struct Card{   //structure Card to store values and suits of cards
   int value;
   char suit;
};

class Hand{
private:
   Card cards[5];   //stores cards of one hand
   int type;    //integer for determining the type of the cards' combination
   bool flush, straight, four, full_house, three, two_pairs, two;    //variables to determine whether each combination true or not
   int counter, max_counter, max_counter2, card_value, card_value2; //variables for finding number of same cards in one hand
public:
   int combination_value, combination_value2, highest[5];    //variables to store highest cards, later used for comparison
   Hand();    //default constructor
   Hand(string (&x)[5]){     //constructor that we use (receive the data as strings)
      for (int i = 0; i < 5; i++){   //converting all values into numbers
         if (x[i][0] == 'T')
            cards[i].value = 10;
         else if (x[i][0] == 'J')
            cards[i].value = 11;
         else if (x[i][0] == 'Q')
            cards[i].value = 12;
         else if (x[i][0] == 'K')
            cards[i].value = 13;
         else if (x[i][0] == 'A')
            cards[i].value = 14;
         else
            cards[i].value = x[i][0] - 48;
         cards[i].suit = x[i][1];
      }
   }
   void DetermineType();    //function for determining type of a hand
   void sort_values(){     //function for sorting the values of cards in one hand
      for (int i = 0; i < 5; i++){
         for (int j = i; j < 5; j++){
            if (cards[i].value > cards[j].value){
                int temp;
                temp = cards[i].value;
                cards[i].value = cards[j].value;
                cards[j].value = temp;
            }
        }
    }
}
int compare(Hand B){    //function for comparing the cards of two hands
    if (type < B.type){     //if Black hand is better then return 1
        return 1;
    }
    else if (type > B.type){   //if White hand is better then return -1
        return -1;
    }
    else{      //else considers possibilities
        if (type == 1 || type == 4 || type == 5 || type == 9){
        //Straight flush, Flush, Straight, Highest card
            for (int t = 0; t < 5; t++){
                if (highest[t] > B.highest[t])   //if Black's highest card is larger return 1
                    return 1;
                else if (highest[t] < B.highest[t])   //if White's highest card is larger return -1
                    return -1;
            }
            return 0;   //if tie, return 0
        }
        else if (type == 2 || type == 3 || type == 6){
        //Four of a kind, Full house, Three of a kind
            if (combination_value > B.combination_value)   //if Black's value of combination is larger, return 1
                return 1;
            else if (combination_value < B.combination_value)   //if White's value of combination is larger, return -1
                return -1;   //if tie, return 0
            return 0;
        }
        else if (type == 7){    //Two pairs
            if (combination_value > B.combination_value)   //if Black's value of first combination is larger, return 1
                return 1;
            else if (combination_value < B.combination_value) //if White's value of first combination is larger, return -1
                return -1;
            else if (combination_value2 > B.combination_value2) //if Black's value of second combination is larger, return 1
                return 1;
            else if (combination_value2 < B.combination_value2) //if White's value of second combination is larger, return -1
                return -1;
            else if (highest[0] > B.highest[0])  //if Black's value of highest card is larger, return 1
                return 1;
            else if (highest[0] < B.highest[0])  //if White's value of highest card is larger, return -1
                return -1;
            return 0;   //if tie, return 0
        }
        else if (type == 8){   //Pair
            if (combination_value > B.combination_value) //if Black's value of combination is larger, return 1
                return 1;
            else if (combination_value < B.combination_value) //if White's value of combination is larger, return -1
                return -1;
            //Then considering the possibilities of next three highest cards
            else if (highest[0] > B.highest[0])
                return 1;
            else if (highest[0] < B.highest[0])
```

```
                return -1;
            else if (highest[1] > B.highest[1])
                return 1;
            else if (highest[1] < B.highest[1])
                return -1;
            else if (highest[2] > B.highest[2])
                return 1;
            else if (highest[2] < B.highest[2])
                return -1;
            return 0;    //if tie, return 0
        }
    }
    return 0;
  }
};

void Hand::DetermineType(){    //function for determining the type of the hand
  counter = 0, max_counter = 0, max_counter2 = 0, card_value = 0, card_value2 = 0;
  sort_values();    //for convenience firstly sort the cards' values
  for (int i = 0; i < 5; i++){   //counting how many same cards we have and storing their values
    counter = 0;
    for (int j = 0; j < 5; j++){
      if (cards[i].value == cards[j].value)
        counter += 1;
    }
    if (counter >= max_counter){
      max_counter = counter;
      card_value = cards[i].value;
    }
  }
  for (int i = 0; i < 5; i++){   //counting how many same cards other than "first same cards" we have and storing their values
    counter = 0;
    for (int j = 0; j < 5; j++){
      if (cards[i].value == cards[j].value && cards[i].value != card_value)
        counter += 1;
    }
    if (counter >= max_counter2){
      max_counter2 = counter;
      card_value2 = cards[i].value;
    }
  }
  //by using the number of same cards and their values find the combination of the hand
  four = false;
  if (max_counter == 4)
    four = true;
  full_house = false;
  if (max_counter == 3 && max_counter2 == 2)
    full_house = true;
  flush = true;
  for (int i = 0; i < 4; i++){
    if (cards[i].suit != cards[i + 1].suit)
      flush = false;
  }
  straight = true;
  for (int i = 0; i < 4; i++){
    if ((cards[i].value + 1) != cards[i + 1].value)
      straight = false;
  }
```

```
    three = false;
    if (max_counter == 3)
        three = true;
    two_pairs = false;
    if (max_counter == 2 && max_counter2 == 2)
        two_pairs = true;
    two = false;
    if (max_counter == 2)
        two = true;
    for (int e = 0; e < 5; e++){
        highest[e] = cards[4 - e].value;
    }
    combination_value = card_value;    //assigning the cards values of same cards to combination values
    combination_value2 = card_value2;
    //then we assign the type for each possible combination, and if needed combination values and highest cards values
    if (flush && straight)
        type = 1;
    else if (four)
        type = 2;
    else if (full_house)
        type = 3;
    else if (flush)
        type = 4;
    else if (straight)
        type = 5;
    else if (three)
        type = 6;
    else if (two_pairs){
        type = 7;
        combination_value = card_value;
        combination_value2 = card_value2;
        for (int i = 0; i < 5; i++){
            if (cards[i].value != card_value && cards[i].value != card_value2){
                highest[0] = cards[i].value;
                break;
            }
        }
    }
    else if (two){
        type = 8;
        for (int i = 0; i < 5; i++){
            if (cards[i].value == card_value)
                cards[i].value = 0;
        }
        sort_values();
        highest[0] = cards[4].value;
        highest[1] = cards[3].value;
        highest[2] = cards[2].value;
    }
    else
        type = 9;
}

int main(){
    string cards1[5], cards2[5], str;   //two arrays for hands and str for reading first input
    while (cin >> str){    //while we have some input
        cards1[0] = str;   //this input is the first card of Black's hand
        //receiving the cards of Black's and White's hands
        for (int i = 1; i < 5; i++){
            cin >> cards1[i];
        }
        for (int i = 0; i < 5; i++){
            cin >> cards2[i];
        }
        //A is Black and B is White
        Hand A(cards1), B(cards2);   //passing values to Hand constructor
        A.DetermineType();   //determining Black's hand type
        B.DetermineType();   //determining White's hand type
        int result = A.compare(B); //comparing the hands type
        if (result == 1){    //if result is 1 then Black has better cards
            cout << "Black wins." << endl;
        }
        else if (result == -1){ //if result is -1 then White has better cards
            cout << "White wins." << endl;
        }
        else{    //if result is 0 then tie
            cout << "Tie." << endl;
        }
    }
    return 0;
}
```

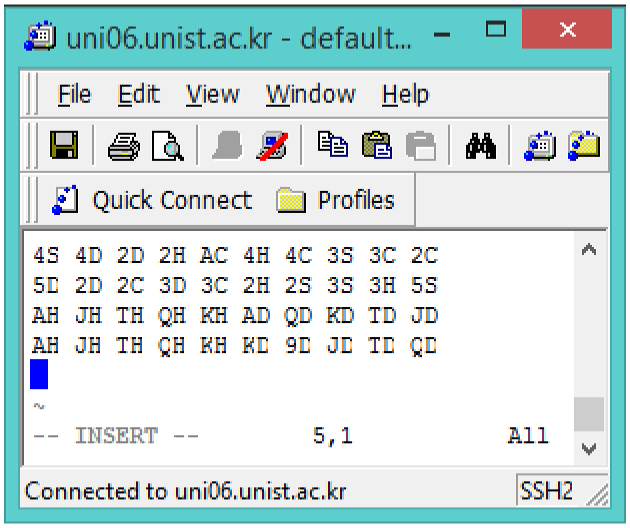
3. Sample input

First sample input


uni06.unist.ac.kr - defaul...
File Edit View Window Help
Quick Connect Profiles
Connected to uni06.unist.ac.kr
SSH2

```
2H 3D 5S 9C KD 2C 3H 4S 8C AH
2H 4S 4C 2D 4H 2S 8S AS QS 3S
2H 3D 5S 9C KD 2C 3H 4S 8C KH
2H 3D 5S 9C KD 2D 3H 5C 9S KH

-- INSERT --        5,1          All
```

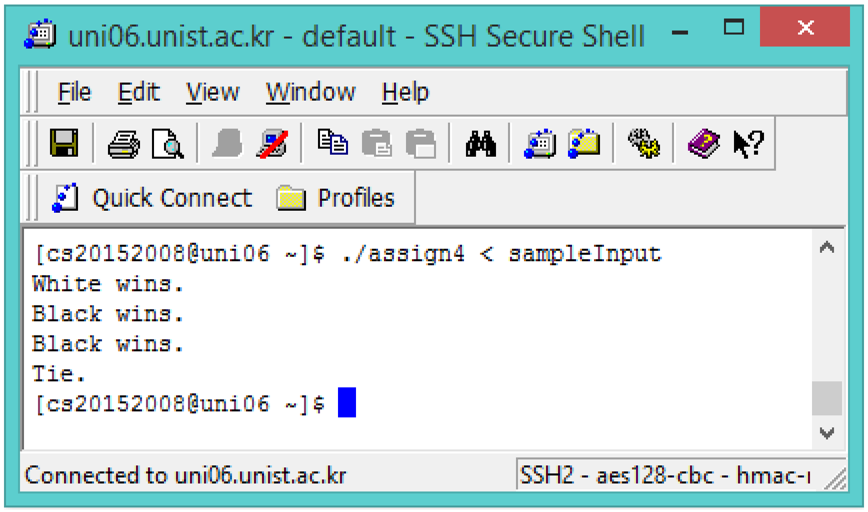
Second sample input


uni06.unist.ac.kr - default...
File Edit View Window Help
Quick Connect Profiles
Connected to uni06.unist.ac.kr
SSH2

```
4S 4D 2D 2H AC 4H 4C 3S 3C 2C
5D 2D 2C 3D 3C 2H 2S 3S 3H 5S
AH JH TH QH KH AD QD KD TD JD
AH JH TH QH KH KD 9D JD TD QD

~
-- INSERT --         5,1          All
```

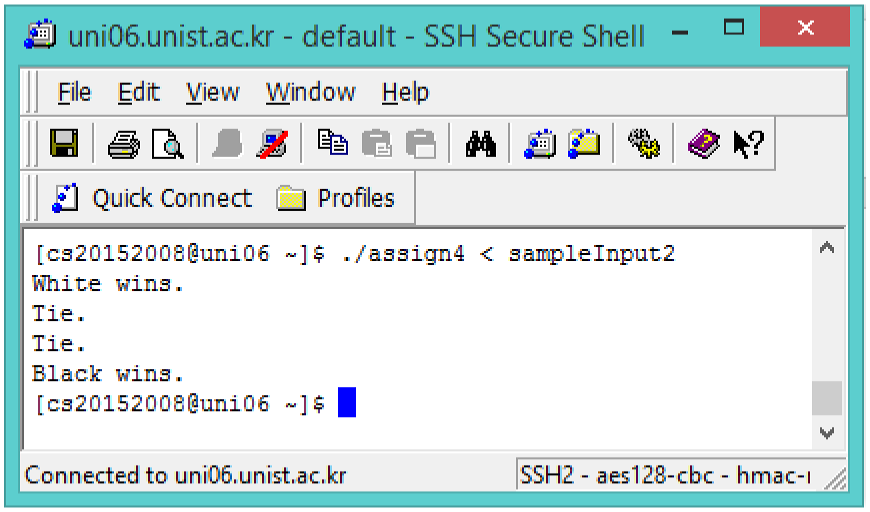
4. Sample output

First sample output


uni06.unist.ac.kr - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
Connected to uni06.unist.ac.kr
SSH2 - aes128-cbc - hmac-

```
[cs20152008@uni06 ~]$ ./assign4 < sampleInput
White wins.
Black wins.
Black wins.
Tie.
[cs20152008@uni06 ~]$
```

Second sample output


uni06.unist.ac.kr - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
Connected to uni06.unist.ac.kr
SSH2 - aes128-cbc - hmac-

```
[cs20152008@uni06 ~]$ ./assign4 < sampleInput2
White wins.
Tie.
Tie.
Black wins.
[cs20152008@uni06 ~]$
```