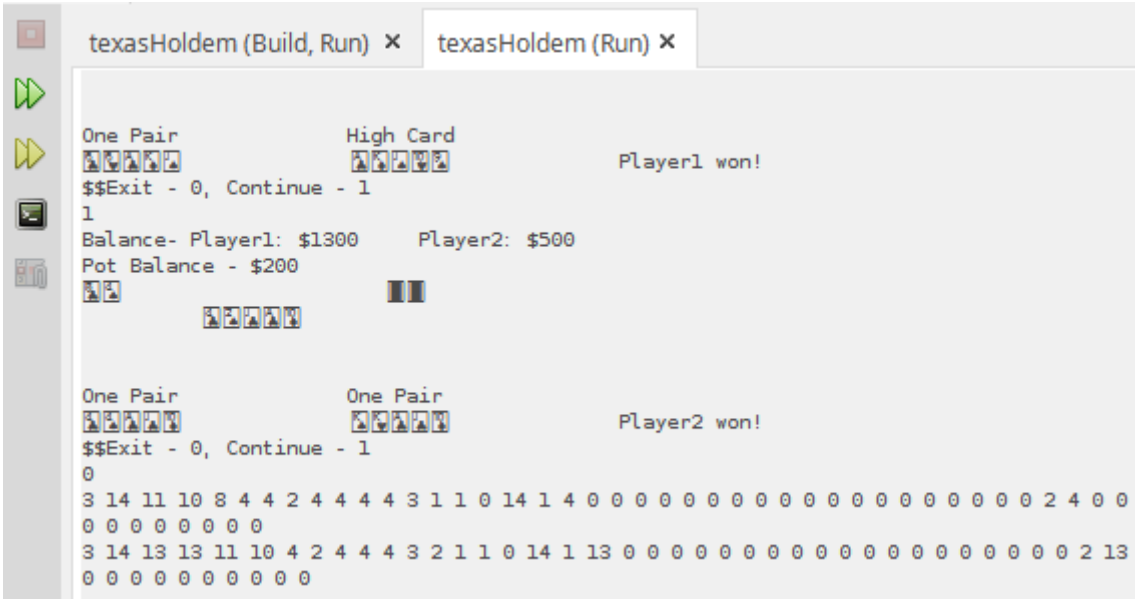
Project 1
<Texas Holdem>



CSC 44083
Name: Lee, Byoung Mo
Data: 4/14/2019

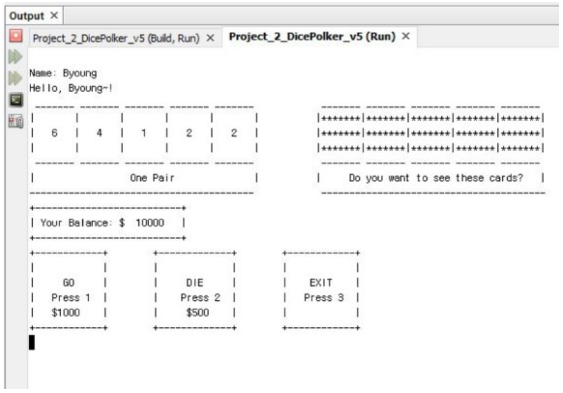
1. Purpose of the project

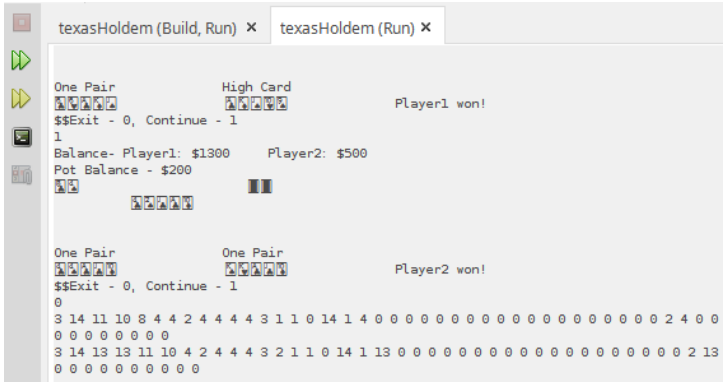
(1) Make my favorite board game, Texas Holdem with C++

(2) Use the tools we learned in the 17A class so far as follow,

- Memory allocation with arrays and structures
- Functions with structures, used as input and output to the function
- Pointers with arrays and arrays of structures, internally as well as externally
- Use of character arrays as well as string objects
- Reading and writing to binary files

(3) Make more improved poker program than CIS5 which was dice poker





(4) Research the current technology for poker AI, especially betting strategy

(5) Try to make a program having a simple betting strategy

- Only consider a ‘Heads-Up’ which is only for two players
- As there are maximum 4 betting chances in the game originally, try simply to make minimum 2 betting chances which are in the middle such as after ‘Flop’, 3 cards in the community and 2 cards for an each player and at the final moment such as after ‘Showdown’, 5 cards in the community and 2 cards for an each player.

(* Although I had an ambitious plan for this project, I spent most of my time to make functions to display each hands with sorted cards, 5 cards of 7 and was not able to make any betting solution due to the limited time and my knowledge. However, I learned a lot by this project. First, I must not code without detailed plans with flow chart or pseudo code because these works enable me to decrease errors and wasting time. Second, important factor to make a good program is not just a skill of coding but a deep critical thinking about procedures.)

2. Flow Chart of ‘main’ function – Attachment #1

3. Structure

(1) Cards

Name	Variables	Type	Description
cards	faces	Face (enum)	Two(0) ~ Ace(12)
	suits	Suit (enum)	Spades(0) ~ Clubs(3)
	print	String	52 Cards (Unicode Standard 12.0) 1F0A1 ~ 1F0DE

(2) Players

Name	Variables	Type	Description
player[2]	name[20]	char	Player1, Player2
	blind	bool	For mark small & big blind (but, actually I didn't use this variable because there is no betting procedure in this program)
	balance	int	Each player's balance
	mycards	Cards*	Each player's face, suit, print (The order of this array should be same as original in the process of dealing. So, I copied this structure in the function for sorting)
	hands	int*	For each betting chance, this array have all the current information for player to decide how to bet at that moment

4. Key Variable - player->hands[50]

- An array of int in the player structure

- This information is updated at every betting chance with getMyCardInfo function.

location	description
0	Betting #1st ~ 4^{th} (0 ~ 3) Since there are different factors for each betting moment, I put this one to the first location.
1-7	Faces (2~14)
8-15	Suits (1~4)
16	High Card

17	Pairs (0 – no pair, 1 – one pair, 2 – two pairs, 3 – two pairs or up)
18 ~22	Faces for pairs
23	One Three of a cards, then hands[23]=1, Two Three of a Cards, then 2
24~25	Faces for Three of a card
26	Four of a card, then hands[24]=1
27	Face of Four of a card
28	Full House, then hands[26]=1
29	For the Full House, a face of the three of a card
30	For the Full House, a face of the pair
31	Flush, then hands[31]=1
32	The suit of the flush
33	Straight, then hands[33]=1
34	The highest face of the straight (*A-2-3-4-5 and 10-J-Q-K-A)
35	Straight Flush, then hands[35]=1
36	The highest face of the straight flush
37	Suit of the straight flush
38	Final hands

5. Codes

```
/*
 * File:   main.cpp
 * Author: Byoung Mo Lee
 * Created on April 7, 2019 22:58 PM
 * Purpose: Texas Holdem
 *
 */

//System Libraries Here
#include <iostream>//I/O Library
#include <cstdlib>   //Srand
#include <ctime>     //Time to set random number seed
#include <cmath>     //Math Library
#include <fstream> //file I/O Library
#include <cctype>  //ch+10
#include <string>  //string library
#include <clocale> //print card library

using namespace std;

//User Libraries Here
//Global Constants Only, No Global Variables
//Like PI, e, Gravity, or conversions
//Structure
enum Face {Two, Three, Four,Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King, Ace};
enum Suit {Spades,Hearts, Diamonds, Clubs};
```

```
enum Hand {HighCard=1,OnePair,TwoPair,  ThreeOfAKind, Straight, Flush,FullHouse,
FourOfAKind,StraightFlush };

struct Cards{
   Face faces;
   Suit suits;
   string print;
};

struct Players{
   char name[20];
   bool blind;
   int balance;
   Cards* mycards;
   int* hands;
};

//Function Prototypes Here
void  shuffle(Cards* ,int);
int contribute(Players* );
void dealPreflop(Cards* , Players*);
void dealFlop(Cards*,Players*);
void turn(Cards* ,Players* );
void river(Cards* ,Players* );
void getMycardInfo(Cards*,Players*);
void selectionSort(int* ,int,  int );
void selectionSort(Cards* , int , int );
void selectionSortS(Cards* , int , int );
void swap(int &, int &);
void checkPairs(int* , int , int );
void clearEval(int* , int , int );
void checkFullHouse(int* array);
void checkFlush(int* ,int, int);
void checkStraight(int* ,int, int);
void checkStraightFlush(Players*);
void showHands(int* );
void showCards(Players* p);
int binarySearch(Cards* , int, int , int );
Cards* sortOnePair(Cards*,int );
Cards* sortTwoPair(Cards* ,int ,int );
Cards* sortTriple(Cards* ,int );
void displayCards(Cards* ,int* );
Cards* sortStraight(Cards* ,int );
Cards* sortFlush(Cards* ,int);
Cards* sortFullHouse(Cards* ,int ,int );
int decideWinner(Players* );
void calculateBalance(Players*,int, int);
void writeBinaryFile(int* ,fstream &, string );

void displayFile(string , int* , int );

//Execution begins here



int main(int argc, char** argv) {
   //Set the random number seed
```

```
//Declare Variables
fstream dataFile;
fstream handsInfo;

const int NUM_FACES=13;
const int NUM_SUITS=4;
const int SIZE=52;
Cards deckOfCards[SIZE];
int choice=0;
int pot;
int winner;
int cont;



Players player[2]={{"me",0,1000,NULL,NULL},{"computer",1,1000,NULL,NULL}};

dataFile.open("hands.txt",ios::out | ios::app);

do{
   pot=0;
   winner=-1;
   cont=1;

//Initialize or input i.e. set variable values

player[0].mycards=new Cards[7];
player[1].mycards=new Cards[7];
player[0].hands=new int [50];
player[1].hands=new int [50];

for(int i=0;i<NUM_FACES;i++){
    for(int j=0;j<NUM_SUITS;j++){
         deckOfCards[j*13+i].faces=static_cast<Face>(i);
         deckOfCards[j*13+i].suits=static_cast<Suit>(j);
         if(j*13+i==12)    deckOfCards[j*13+i].print="\U0001F0A1";
         else if(j*13+i==0)    deckOfCards[j*13+i].print="\U0001F0A2";
         else if(j*13+i==1)    deckOfCards[j*13+i].print="\U0001F0A3";
         else if(j*13+i==2)    deckOfCards[j*13+i].print="\U0001F0A4";
         else if(j*13+i==3)    deckOfCards[j*13+i].print="\U0001F0A5";
         else if(j*13+i==4)    deckOfCards[j*13+i].print="\U0001F0A6";
         else if(j*13+i==5)    deckOfCards[j*13+i].print="\U0001F0A7";
         else if(j*13+i==6)    deckOfCards[j*13+i].print="\U0001F0A8";
         else if(j*13+i==7)    deckOfCards[j*13+i].print="\U0001F0A9";
         else if(j*13+i==8)    deckOfCards[j*13+i].print="\U0001F0AA";
         else if(j*13+i==9)    deckOfCards[j*13+i].print="\U0001F0AB";
         else if(j*13+i==10)    deckOfCards[j*13+i].print="\U0001F0AD";
         else if(j*13+i==11)    deckOfCards[j*13+i].print="\U0001F0AE";
         else if(j*13+i==25)    deckOfCards[j*13+i].print="\U0001F0B1";
         else if(j*13+i==13)    deckOfCards[j*13+i].print="\U0001F0B2";
         else if(j*13+i==14)    deckOfCards[j*13+i].print="\U0001F0B3";
         else if(j*13+i==15)    deckOfCards[j*13+i].print="\U0001F0B4";
         else if(j*13+i==16)    deckOfCards[j*13+i].print="\U0001F0B5";
         else if(j*13+i==17)    deckOfCards[j*13+i].print="\U0001F0B6";
         else if(j*13+i==18)    deckOfCards[j*13+i].print="\U0001F0B7";
         else if(j*13+i==19)    deckOfCards[j*13+i].print="\U0001F0B8";
```

```
            else if(j*13+i==20)   deckOfCards[j*13+i].print="\U0001F0B9";
            else if(j*13+i==21)   deckOfCards[j*13+i].print="\U0001F0BA";
            else if(j*13+i==22)   deckOfCards[j*13+i].print="\U0001F0BB";
            else if(j*13+i==23)   deckOfCards[j*13+i].print="\U0001F0BD";
            else if(j*13+i==24)   deckOfCards[j*13+i].print="\U0001F0BE";
            else if(j*13+i==38)   deckOfCards[j*13+i].print="\U0001F0C1";
            else if(j*13+i==26)   deckOfCards[j*13+i].print="\U0001F0C2";
            else if(j*13+i==27)   deckOfCards[j*13+i].print="\U0001F0C3";
            else if(j*13+i==28)   deckOfCards[j*13+i].print="\U0001F0C4";
            else if(j*13+i==29)   deckOfCards[j*13+i].print="\U0001F0C5";
            else if(j*13+i==30)   deckOfCards[j*13+i].print="\U0001F0C6";
            else if(j*13+i==31)   deckOfCards[j*13+i].print="\U0001F0C7";
            else if(j*13+i==32)   deckOfCards[j*13+i].print="\U0001F0C8";
            else if(j*13+i==33)   deckOfCards[j*13+i].print="\U0001F0C9";
            else if(j*13+i==34)   deckOfCards[j*13+i].print="\U0001F0CA";
            else if(j*13+i==35)   deckOfCards[j*13+i].print="\U0001F0CB";
            else if(j*13+i==36)   deckOfCards[j*13+i].print="\U0001F0CD";
            else if(j*13+i==37)   deckOfCards[j*13+i].print="\U0001F0CE";
            else if(j*13+i==51)   deckOfCards[j*13+i].print="\U0001F0D1";
            else if(j*13+i==39)   deckOfCards[j*13+i].print="\U0001F0D2";
            else if(j*13+i==40)   deckOfCards[j*13+i].print="\U0001F0D3";
            else if(j*13+i==41)   deckOfCards[j*13+i].print="\U0001F0D4";
            else if(j*13+i==42)   deckOfCards[j*13+i].print="\U0001F0D5";
            else if(j*13+i==43)   deckOfCards[j*13+i].print="\U0001F0D6";
            else if(j*13+i==44)   deckOfCards[j*13+i].print="\U0001F0D7";
            else if(j*13+i==45)   deckOfCards[j*13+i].print="\U0001F0D8";
            else if(j*13+i==46)   deckOfCards[j*13+i].print="\U0001F0D9";
            else if(j*13+i==47)   deckOfCards[j*13+i].print="\U0001F0DA";
            else if(j*13+i==48)   deckOfCards[j*13+i].print="\U0001F0DB";
            else if(j*13+i==49)   deckOfCards[j*13+i].print="\U0001F0DD";
            else if(j*13+i==50)   deckOfCards[j*13+i].print="\U0001F0DE";
            }
}
shuffle(deckOfCards,SIZE);
pot=contribute(player);
dealPreflop(deckOfCards, player);
getMycardInfo(deckOfCards,player);
dealFlop(deckOfCards,player);
getMycardInfo(deckOfCards,player);
turn(deckOfCards,player);
getMycardInfo(deckOfCards,player);
river(deckOfCards,player);
getMycardInfo(deckOfCards,player);
showCards(player);
winner=decideWinner(player);
calculateBalance(player,winner,pot);

for(int i=0;i<2;i++){
   for(int j=0;j<7;j++){
   dataFile << player[i].mycards[j].print;
   }
   dataFile << "$";
   dataFile << player[i].hands[38] << "$";
}
dataFile << winner << endl;

for(int i=0;i<2;i++){
```

```
        writeBinaryFile(player[i].hands, handsInfo, "handsInfo.dat");
    }

    cout << "Exit - 0, Continue - 1 " << endl;
    cin >> choice;
    for(int i=0;i<2;i++){
        delete [] player[i].mycards;
        delete [] player[i].hands;
    }

    }while(choice!=0); //&& b[2]!=0);


    for(int i=0;i<2;i++){
        displayFile("handsInfo.dat", player[i].hands, 50);
    }
    dataFile.close();

    return 0;
}

void  shuffle(Cards* a,int n){
    // Initialize seed randomly
    srand(time(0));

    for (int i=0; i<n ;i++)
    {
        // Random for remaining positions.
        int r = i + (rand() % (n -i));

        swap(a[i], a[r]);
     }
}

 int contribute(Players* a){
     int aa=100;
     int bb=100;

     if(a[0].blind==0){
         a[0].balance-=aa;
         a[1].balance-=bb;
     }
     else {
        a[0].balance-=bb;
        a[1].balance-=aa;
     }

     cout << "Balance- Player1: $" << a[0].balance << "      Player2: $" << a[1].balance << endl;
     cout << "Pot Balance - $" << aa+bb << endl;

     return aa+bb;
    }

 void dealPreflop(Cards* a, Players* b){  //deal total two card but one card each time

     if(b[0].blind==0){
         b[0].mycards[0]=a[0];
```

```
        b[0].mycards[1]=a[2];
        b[1].mycards[0]=a[1];
        b[1].mycards[1]=a[3];
    }
    else{
        b[1].mycards[0]=a[0];
        b[1].mycards[1]=a[2];
        b[0].mycards[0]=a[1];
        b[0].mycards[1]=a[3];
    }
    cout << b[0].mycards[0].print << b[0].mycards[1].print ;
    cout << "                    " << "\U0001F0A0" << "\U0001F0A0" << endl;

    b[0].hands[0]=0;              // for decision making of betting
    b[1].hands[0]=0;              // 0 means the 1st betting stage
}

void dealFlop(Cards* a,Players* b){
    cout << "          " << a[5].print << a[6].print  << a[7].print ;  //before flopping, dealer burn a card

    b[0].mycards[2]=a[5];
    b[0].mycards[3]=a[6];
    b[0].mycards[4]=a[7];
    b[1].mycards[2]=a[5];
    b[1].mycards[3]=a[6];
    b[1].mycards[4]=a[7];

    b[0].hands[0]=1;
    b[1].hands[0]=1;           // 1 means the 2nd betting stage
}

void turn(Cards* a,Players* b){    //before turning, dealer burn a card
    cout << a[9].print;

    b[0].mycards[5]=a[9];
    b[1].mycards[5]=a[9];

    b[0].hands[0]=2;       //2 means the 3rd betting stage
    b[1].hands[0]=2;
}

 void river(Cards* a,Players* b){
    cout << a[11].print << endl << endl << endl;       //before revering, dealer burn a card

    b[0].mycards[6]=a[11];
    b[1].mycards[6]=a[11];

    b[0].hands[0]=3;
    b[1].hands[0]=3;       //3 means the 4th betting stage
}


void getMycardInfo(Cards* a,Players* b){
    int stage=0;
    if(b[0].hands[0]==0) stage=2;
    else if(b[0].hands[0]==1) stage=5;
    else if(b[0].hands[0]==2) stage=6;
```

```
    else if(b[0].hands[0]==3) stage=7;

    for(int j=0;j<2;j++){
        for(int i=0;i<stage;i++){
        b[j].hands[i+1]=b[j].mycards[i].faces+2;   //player.hands array[1]~[7]:faces
        b[j].hands[i+8]=b[j].mycards[i].suits+1;   //player.hands array[8]~[15]:suits
        //cout << b[j].hands[i+1] << endl;
        //cout << b[j].hands[i+8] << endl;
        }
    }
    for(int i=0;i<2;i++){
      selectionSort(b[i].hands,1,stage);
      selectionSort(b[i].hands,8,stage);
    }

    for(int i=0;i<2;i++){
      checkPairs(b[i].hands,17,stage);
      checkFullHouse(b[i].hands);
      checkFlush(b[i].hands,8,stage);
      checkStraight(b[i].hands,1,stage);
      checkStraightFlush(b);
    }

    //for(int j=0;j<2;j++){
    //   for(int i=0;i<40;i++){
    //        cout << b[j].hands[i] << " ";
    //   }
    //  cout << "||||||||||" << endl;
    //}
}

void selectionSort(int* array, int begin, int size){
    int minIndex, maxValue;
    int end=begin+size;
    for(int start=begin;start<end-1;start++){
        minIndex=start;
        maxValue=array[start];
        for(int index=start+1;index<end;index++){
            if(array[index]>maxValue){
                maxValue=array[index];
                minIndex=index;
            }
        }
        swap(array[minIndex],array[start]);
    }
}

void swap(int &a, int &b){
    int temp=a;
    a=b;
    b=temp;
}

void checkPairs(int* array, int begin, int size){
    clearEval(array,begin,50);
    array[16]=array[1];
```

```
   int f=18;
   int count=0;

   for(int i=1;i<size;i++){
      if((array[i]-array[i+1])==0){
         array[begin]+=1;
         array[f+count]=array[i];
         count++;
      }
   }

   if(count>=2){                          //triple or four of a card check
      for(int i=f;i<f+count-1;i++){
         if(array[i]-array[i+1]==0){
            array[begin]-=2;
            if(i+1<=f+count-1){
               if(array[i+1]-array[i+2]==0){
                  array[26]=1;          //mark four of a card
                  array[27]=array[i];  //put the information of face
                  array[i]=0;          //remove pair marking, since it
                  array[i+1]=0;        //moved to four of a card already
                  array[i+2]=0;
               }
               else {
                  array[23]+=1;          //since two triple can happen
                  if(array[23]==2)    array[25]=array[i];
                  else              array[24]=array[i];
                  array[i]=0;
                  array[i+1]=0;
               }
            }
         }
      }
   }


}


void clearEval(int* array, int begin, int size){
   for(int i=begin;i<size;i++){
      array[i]=0;
   }
}

void checkFullHouse(int* array){
   if(array[23]>=1&&array[17]>=1) {
      array[28]=1;
      if(array[24]){
         array[29]=array[24];
         array[23]-=1;
         array[24]=0;
      }
      else{
         array[29]=array[25];
         array[23]-=1;
         array[25]=0;
```

```
        }
        if(array[18]){
            array[30]=array[18];
            array[17]-=1;
            array[18]=0;
        }
        else if(array[19]){
            array[30]=array[19];
            array[17]-=1;
            array[19]=0;
        }
        else if(array[20]){
            array[30]=array[20];
            array[17]-=1;
            array[20]=0;
        }
        else if(array[21]){
            array[30]=array[21];
            array[17]-=1;
            array[21]=0;
        }
        else if(array[22]){
            array[30]=array[22];
            array[17]-=1;
            array[22]=0;
        }
    }
}

void checkFlush(int* array,int begin, int size){

    int index=begin;
    int count=0;
    selectionSort(array,begin,size);
    while(index<begin+5 && count<4){
        if(array[index]-array[index+1]==0) count++;
        else count=0;
        index++;
    }

    if(count>=4){
        array[31]=1;
        array[32]=array[10];  //if Flush, then 3rd card should be the suits in the sorted situation
    }
}

void checkStraight(int* array, int begin, int size){
    int end=begin+size;
    for(int i=0;i<2;i++){
      selectionSort(array,begin,size);
    }
    int count=0;
    int minValue=0;
    int ace=0;
    if(array[begin]==14) ace=1;       //if there is Ace in players.cards, then 1

   for(int i=begin;i<end;i++){
```

```
      if(array[i]-array[i+1]==1) {
         minValue=array[i+1];
         count++;

         if(array[i+1]==0 && count>=3 && ace==1) {  //array[i+1]==0 Two
            ace=2;                  //if A,2,3,4,5, then ace=2
            count++;
         }

         if(count==4) i=end;
      }
      else count=0;
   }
   if(count==4){
      array[33]=1;
      if(ace==2) array[34]=5;   //Two=0, Five=3, Ace=12
      else   array[34]=minValue+4;
    }
}
void checkStraightFlush(Players* a){
    Players* copy=NULL;
    int sCount=0;              //straight count
    int fCount=0;              //flush count
    int end=0;                 //the address of end of straight
    copy=a;
    int* array=copy->hands;

    if(array[33]&&array[31]){
       Players* copy=NULL;
       int sCount=0;              //straight count
       int fCount=0;              //flush count
       int end=0;                 //the address of end of straight
       copy=a;

       selectionSort(copy->mycards,0,7);

       for(int i=0;i<7;i++){
          if(copy->mycards[i].faces-copy->mycards[i+1].faces==1){
             sCount++;
             end=i+1;
          }
       }
       for(int i=end;i>end-sCount;i--){
          if(copy->mycards[i].suits-copy->mycards[i-1].suits==0) fCount++;
       }
       if(fCount>=4){
          a->hands[35]=1;          // mark StraightFlush = True
          a->hands[36]=array[34];      // put high card of straight
          a->hands[37]=array[32];      // put suit of flush
          a->hands[33]=0;          // remove information in Straight field
          a->hands[34]=0;
          a->hands[31]=0;          // remove information in flush field
          a->hands[32]=0;
       }

    }
}
```

```
void selectionSort(Cards* c, int begin, int size){
   int minIndex, maxValue;
   int end=begin+size;
   for(int start=begin;start<end-1;start++){
      minIndex=start;
      maxValue=c[start].faces;
      for(int index=start+1;index<end;index++){
         if(c[index].faces>maxValue){
            maxValue=c[index].faces;
            minIndex=index;
         }
      }
      swap(c[minIndex].faces,c[start].faces);
      swap(c[minIndex].suits,c[start].suits);
      swap(c[minIndex].print,c[start].print);
   }
}

void selectionSortS(Cards* c, int begin, int size){
   int minIndex, maxValue;
   int end=begin+size;
   for(int start=begin;start<end-1;start++){
      minIndex=start;
      maxValue=c[start].suits;
      for(int index=start+1;index<end;index++){
         if(c[index].suits>maxValue){
            maxValue=c[index].suits;
            minIndex=index;
         }
      }
      swap(c[minIndex].faces,c[start].faces);
      swap(c[minIndex].suits,c[start].suits);
      swap(c[minIndex].print,c[start].print);
   }
}

void showHands(int* p){

   if(p[35]==1) {
      p[38]= StraightFlush;
      p[39]=p[34];
      cout << "Straight Flush";
   }
   else if(p[26]==1){
      p[38]= FourOfAKind;
      p[39]=p[27];
      cout << "Four Of A Kind";
   }
   else if(p[28]==1){
      p[38]= FullHouse;
      p[39]=p[29];
      p[40]=p[30];
      cout << "Full House";
   }
   else if(p[31]==1){
      p[38]= Flush;
```

```
        p[41]=p[32];
        cout << "Flush";
    }
    else if(p[33]==1) {
        p[38]= Straight;
        p[39]=p[34];
        cout << "Straight";
    }
    else if(p[23]) {
        p[38]= ThreeOfAKind;
        p[39]=p[24];
        cout <<"Three Of A Kind";
    }
    else if(p[17]>=2) {
        p[38]= TwoPair;
        p[39]=p[18];
        p[40]=p[19];
        cout << "Two Pair";
    }
    else if(p[17]==1) {
        p[38]= OnePair;
        p[39]=p[18];
        cout << "One Pair";
    }
    else if(p[17]==0) {
        p[38]= HighCard;
        p[39]=p[16];
        cout << "High Card";
    }

}


void showCards(Players* p){


   for(int i=0;i<2;i++){
      showHands(p[i].hands);
      cout << "              ";
   }
   cout << endl;
   for(int i=0;i<2;i++){
      displayCards(p[i].mycards,p[i].hands);
      cout << "              ";
   }
}

void displayCards(Cards* c,int* array){
    Cards* cCopy=NULL;
    cCopy=new Cards[7];

    for(int i=0;i<7;i++){
        cCopy[i].faces=c[i].faces;
        cCopy[i].suits=c[i].suits;
        cCopy[i].print=c[i].print;
    }
```

```
    selectionSort(cCopy, 0, 7);

    if(array[38]==OnePair) cCopy=sortOnePair(cCopy,array[39]-2);
    else if(array[38]==TwoPair) cCopy=sortTwoPair(cCopy,array[39]-2,array[40]-2);
    else if(array[38]==ThreeOfAKind) cCopy=sortTriple(cCopy,array[39]-2);
    else if(array[38]==Straight)   cCopy=sortStraight(cCopy,array[39]-2);
    else if(array[38]==Flush)   cCopy=sortFlush(cCopy,array[41]-1);
    else if(array[38]==FullHouse)   cCopy=sortFullHouse(cCopy,array[39]-2,array[40]-2);

    for(int i=0;i<5;i++){
       cout << cCopy[i].print;
    }
    //cout << endl;
    //for(int i=0;i<7;i++){
    //    cout << c[i].print;
    //}
    //delete[] cCopy;

}

 Cards* sortOnePair(Cards* c,int a){
   int loc=0;

   for(int i=0;i<2;i++){
      loc=binarySearch(c,i,7-i,a);
      if(loc!=i && c[loc].faces!=c[i].faces && loc!=-1){
         swap(c[loc].faces,c[i].faces);
         swap(c[loc].suits,c[i].suits);
         swap(c[loc].print,c[i].print);
      }
      else if(loc==-1)    {
         cout << "i=" << i  <<"   a=" << a << "   loc=" << loc << "   error: loc=-1" << endl;
      }
      selectionSort(c, i+1, 7-(i+1));
   }
   return c;
 }

Cards* sortTwoPair(Cards* c,int a,int b){
   Cards* cCopy=c;
   int loc=0;
   selectionSort(cCopy, 0, 7);

   for(int i=0;i<2;i++){
      loc=binarySearch(cCopy,i,7-i,a);
      if(cCopy[loc].faces!=cCopy[i].faces && loc!=-1){
         swap(cCopy[loc].faces,cCopy[i].faces);
         swap(cCopy[loc].suits,cCopy[i].suits);
         swap(cCopy[loc].print,cCopy[i].print);
      }
      else if(loc==-1)    {
         cout << "i=" << i  <<"   a=" << a << "   loc=" << loc << "   error: loc=-1" << endl;
      }
      selectionSort(c, i+1, 7-(i+1));
   }

   for(int i=2;i<4;i++){
```

```
        loc=binarySearch(cCopy,i,7-i,b);
        if(loc!=i&&loc!=-1){
            swap(cCopy[loc].faces,cCopy[i].faces);
            swap(cCopy[loc].suits,cCopy[i].suits);
            swap(cCopy[loc].print,cCopy[i].print);
        }
        else if(loc==-1)    {
            cout << "i=" << i  <<"   a=" << a << "   loc=" << loc << "   error: loc=-1" << endl;
        }
        selectionSort(c, i+1, 7-(i+1));
    }
    selectionSort(cCopy, 4, 2);

    return cCopy;
}
 Cards* sortTriple(Cards* c,int a){
    int loc=0;

    for(int i=0;i<3;i++){
        loc=binarySearch(c,i,7-i,a);
        if(loc!=i && c[loc].faces!=c[i].faces && loc!=-1){
            swap(c[loc].faces,c[i].faces);
            swap(c[loc].suits,c[i].suits);
            swap(c[loc].print,c[i].print);
        }
        else if(loc==-1)    {
            cout << "i=" << i  <<"   a=" << a << "   loc=" << loc << "   error: loc=-1" << endl;
        }
        selectionSort(c, i+1, 7-(i+1));
    }
    return c;
 }
 Cards* sortStraight(Cards* c,int a){
     selectionSort(c,0,7);
     int loc=binarySearch(c,0,7,a);
    //cout << "location=" << loc << endl;

     if(a==3){
         for(int i=0;i<4;i++){
            loc=binarySearch(c,i,7-i,a-i);
            if(loc!=i){
                swap(c[loc].faces,c[i].faces);
                swap(c[loc].suits,c[i].suits);
                swap(c[loc].print,c[i].print);
            }
         }
         loc=binarySearch(c,4,3,12);
         swap(c[loc].faces,c[4].faces);
         swap(c[loc].suits,c[4].suits);
         swap(c[loc].print,c[4].print);
     }

     else{

        if(loc>=3) cout << "Error";
        else if(loc==2){
            swap(c[0].faces,c[6].faces);
```

```
            swap(c[0].suits,c[6].suits);
            swap(c[0].print,c[6].print);

            swap(c[1].faces,c[5].faces);
            swap(c[1].suits,c[5].suits);
            swap(c[1].print,c[5].print);

            selectionSort(c,0,5);
        }

        else if(loc==0){
            int count=0;
            while(c[count].faces-c[count+1].faces==1||count<4){
                if(c[count].faces-c[count+1].faces!=1){
                    swap(c[count+1].faces,c[7-(count+1)].faces);
                    swap(c[count+1].suits,c[7-(count+1)].suits);
                    swap(c[count+1].print,c[7-(count+1)].print);
                    selectionSort(c,count+1,7-(count+1));
                }
                else count++;
            }
        }
        else if(loc==1){
            int count=0;
            int end=1;

            swap(c[loc].faces,c[0].faces);
            swap(c[loc].suits,c[0].suits);
            swap(c[loc].print,c[0].print);

            swap(c[loc].faces,c[6].faces);
            swap(c[loc].suits,c[6].suits);
            swap(c[loc].print,c[6].print);

            selectionSort(c,1,5);

            while(c[count].faces-c[count+1].faces==1||count<4){
                if(c[count].faces-c[count+1].faces!=1){
                    swap(c[count+1].faces,c[7-(count+1)].faces);
                    swap(c[count+1].suits,c[7-(count+1)].suits);
                    swap(c[count+1].print,c[7-(count+1)].print);
                    selectionSort(c,count+1,7-(count+1));
                }
                else count++;
            }
        }

    }

    return c;
}

Cards* sortFlush(Cards* c,int b){
    int count=0;
    int index=0;
    selectionSortS(c,0,7);
    while(count<2 && index<5){
```

```
        if(c[index].suits-b==0) index++;
        else{
            swap(c[index].faces,c[6-count].faces);
            swap(c[count].suits,c[6-count].suits);
            swap(c[count].print,c[6-count].print);
            count++;
        }
    }
    selectionSort(c,0,5);
    return c;
}

Cards* sortFullHouse(Cards* c,int a,int b){
    int loc;
    int count=0;
    for(int i=0;i<3;i++){
        loc=binarySearch(c,i,7-i,a);
        if(loc!=i && c[loc].faces!=c[i].faces && loc!=-1){
            swap(c[loc].faces,c[i].faces);
            swap(c[loc].suits,c[i].suits);
            swap(c[loc].print,c[i].print);
        }
        else if(loc==-1)    {
            cout << "i=" << i  <<"   a=" << a << "   loc=" << loc << "   error: loc=-1" << endl;
        }
        selectionSort(c, i+1, 7-(i+1));
    }

    for(int i=3;i<5;i++){
        loc=binarySearch(c,i,7-i,b);
        if(loc!=i && c[loc].faces!=c[i].faces && loc!=-1){
            swap(c[loc].faces,c[i].faces);
            swap(c[loc].suits,c[i].suits);
            swap(c[loc].print,c[i].print);
        }
        else if(loc==-1)    {
            cout << "i=" << i  <<"   a=" << a << "   loc=" << loc << "   error: loc=-1" << endl;
        }
        selectionSort(c, i+1, 7-(i+1));
    }

    return c;
}
 int binarySearch(Cards* c, int beg, int size, int value){
     int end=beg+size;
     int first=beg,
            last=end-1,
            middle,
            position=-1;
     bool found=false;

     while(!found && first <= last){
         middle=(first+last)/2;
         if(c[middle].faces==value){
             found=true;
             position=middle;
         }
```

```
        else if(c[middle].faces<value)   last=middle-1;  //since ascending order
        else   first=middle+1;                          //since ascending order
    }
    return position;
}

int decideWinner(Players* p){
    int winner=-1;   //if player1 won winner=0, player2 won winner=1
    int points[2]={0,0};   //points[0] - points total for player1, points[1]-player2
    int diff=0;

    for(int i=0;i<2;i++){
        if(p[i].hands[38]==Straight||p[i].hands[39]==5){
            points[i]=p[i].hands[16]*1000+12*10+p[i].hands[38];
        }
        else if(p[i].hands[38]==HighCard){
            points[i]=p[i].mycards[0].faces+p[i].mycards[1].faces
                    +p[i].mycards[2].faces+p[i].mycards[3].faces+p[i].mycards[4].faces;
        }
        else{
          points[i]=p[i].hands[38]*1000+p[i].hands[39]*10+p[i].hands[16];
        }
   }

    diff=points[0]-points[1];
    if(diff>0) winner=0;
    else if(diff<0) winner =1;
    else winner=-1;

    cout << "Player" << winner+1 << " won!" << endl;

    return winner;
}

void calculateBalance(Players* p, int a, int pot){
    p[a].balance+=pot;
    if(a==-1) {
        p[0].balance+=pot/2;
        p[1].balance+=pot/2;
    }
}

void writeBinaryFile(int* array,fstream &file, string name){
   file.open(name, ios::out | ios::binary );
   file.write(reinterpret_cast<char*>(array),sizeof(array));
   file.close();
}

void displayFile(string name, int* array, int n){
   fstream file(name, ios::in | ios::binary);
   file.read(reinterpret_cast<char* >(array), sizeof(array));
   while(!file.eof()){
       for(int i=0;i<n;i++){
           cout << array[i] << " ";
       }
       file.read(reinterpret_cast<char* >(array), sizeof(array));
       cout << endl;
```

```
    }
    file.close();
}
```