HO CHI MINH UNIVERSITY OF SCIENCE

Faculty of Information Technology

PROJECT REPORT

PROGRAMMING TECHNIQUES

MIDTERM PROJECT – POINTERS

Group member:

- 20127658 - Nguyễn Ngọc Tuấn

- 20127498 - Nguyễn Văn Hiếu

Theory Teacher:

- Nguyễn Hải Minh

- Phan Thị Phương Uyên

Practice Teacher:

- Bùi Huy Thông

- Dương Nguyễn Thái Bảo

I. General

- Knowing how to play poker simply.

- Improving my knowledge of programming techniques.

II. Project Content

II.1. Card shuffling & Dealing

1. void shuffleCards(int deck[SUITS][FACES]);

- We use random_shuffle(pointer to start, pointer to end) from the <algorithm> library, the function uses some unspecified sources of randomness from program to swap the value of each element with other random picked elements.

2. void printCardsShuffling(int deck[SUITS][FACES], const char* suits[], const char* faces[]);

- By using the variable “count”, we can search for the orders of card in deck that have been shuffled. Each time the current order is found, the variable raised by 1, then using break statement to restart the search from the first element of deck.

II.2. Card game: Five-card Poker

1.a) int dealingForHand(int deck[SUITS][FACES]);**

- We begin manipulating “card_ord” for the function. Singleplayer-mode means they will get their cards from order 0 to 4, each time the function found the order in deck, it stores value to one hand.

1.b) void printHand(int hand, const char* suits[], const char* faces[]);**

- We reuse the printCards function in the preference to print the shape and the space between cards.

1.c) int createHandTest(int deck[SUITS][FACES], int a[]);**

- The program lets the user input 5 cards from deck, to choose the specific number, then it use the printCardsShuffling function to print the current deck (The first card starts from 0).

1.d) int isFourOfAKind(int hand);**

- Because of creating the Sortcard function before, we just have 2 correct form in this categories (rank 1 = rank 2 = rank 3 = rank 4 or rank 2 = rank 3 = rank 4 = rank 5),we don't worry about suits.

1.e) isFullHouse(int hand);**

- Because of creating the Sortcard function, we only have 2 correct form int his categories (rank 1 = rank 2 = rank 3 and rank 4 = rank 5) or (rank1 = rank2 and rank 3 = rank 4 = rank 5).

1.f) isFlush(int hand);**

- In this categories we have some case overlap with Straight so we have to remove it.

1.g) isStraight(int hand);**

- Similar to isFlush function we have to remove flush's case.

1.h) int isStraightFlush(int hand);**

- This category is the intersection of the 2 categories above.

1.i) isThreeOfAKind(int hand);**

- In this function, we have some cases overlapping with the above two categories(Four-of-a-kind or quads and Full-house), so we have to remove 2 categories. Because we had removed 2 duplicate cases and sort card from the beginning, we just check if they have three adjacent cards.

1.j) int isTwoPairs(int hand)**;

- Similar to Three of a kind case , we have some cases overlapping with two categories above (Four of a kind or quads and Full house), so we have to remove 2 categories. With sort card function, we will check 3 case (a,b,a || a,a,b || b,a,a) with “a” is pair card and “b” is another card).

1.k) isPair(int hand);**

- Eliminating 4 categories (Four of a kind or quads, Full house, Three of a kind, Two pair), then checking only if they have 2 card similar rank .

1.h) int getHighestCard(int hand)**;

- Eliminating all categories.

2.a) int* dealingForHands(int deck[SUITS][FACES], int n,int*** hands);**

- Multiplayer mode means the card_ord%n will be stored to n players each round, once card_ord%n = 0 again, the card_ord_person will be raised by 1, mean array hands would be expected to stop at [n-1][4][x].

2.b) int getStatusOfHand(int hand);**

- Return the point after check by function 1.d,e,f,g,h,i,j,k,l .

2.c) int* rankingHands(int* hands, int n);**

- Having 2 arrays called result and ranking, first the result will store all the status point of each player (player i to result[i]), while ranking just store the number of i (the i^{th} player). After stored, we would start the sort of result descending, thus the ranking would be sorted together. After all we will get the output as expected, the highest will be stored in both first elements of ranking and result, an so on.

2.d) int* evaluateHands(int deck[SUITS][FACES], int* hands, int n, int s);**

- The function used loops for s rounds. In each loop, it would:
+ Reset the result[], and hand[].
+ For each hand[], get their status and store to result[].
+ Print out the result.
+ Shuffle deck, then deal again.

- After s rounds, congratulate the final winner.

*** Function Bonus:**

1. void sortCards(int a);**

- After distributing cards for the player, they sorted their five cards in an ascending order (Rank) by the SortCard function:

```
void SortCards(int** a) {
    for (int i = 0; i < 4; i++)
        for (int j = i + 1; j < 5; j++)
            if (a[i][1] > a[j][1])
            {
                int temp_suit = a[j][0];
                int temp_face = a[j][1];
                a[j][0] = a[i][0];
                a[j][1] = a[i][1];
                a[i][0] = temp_suit;
                a[i][1] = temp_face;
            }
}
```

- Simplifying the explaination: I use suits-num and rank-num variables for representing the name suits and name rank for five card like this:

+ Suits 1 - Rank 1
+ Suits 2 - Rank 2
+ Suits 3 - Rank 3
+ Suits 4 - Rank 4
+ Suits 5 - Rank 5

2. int checkFlush(int ** hand);

int checkStraight(int hand);**

- I created two functions to check flush and straight cases, like this:

```
int checkFlush(int** hand) {
    for (int i = 0; i < 3; i++) {
        if (hand[i][0] != hand[i + 1][0])
            return 0;
    }
    return 1;
}

int checkStraight(int** hand) {
    for (int i = 0; i < 3; i++)
        if (hand[i + 1][1] - hand[i][1] != 1)
            return 0;
    return 1;
}
```

3. void PickUpCardBetter(int deck[SUITS][FACES], int*** hands, int n);

- Just understanding in a simple way, if we have 4 players (including the dealer), 5 cards of the dealer will be number 1,5,9,13,17 after distributing the cards in deck. Dealer will be the first player to pick up new card, and that card will be the 21st card in deck beacause we have for player so the card after distributings the cards left is starting 21st. Then we run a loop of 5 old cards to check whether the 21st card replace each of old 5 cards or not, to make dealer in better situation then we finish picking up and replacing that card with old cards. If 3 cards can't make dealer in better situation we replace 3rd card from picking up to replace the first of 5 old card because we had sort card so the first of 5 old cards is the card that has the smallest rank value.

4. void chooseDeck(int deck[SUITS][FACES]);

- In this option, we have 3 choices:

1. For 52 cards in deck.

2. Shuffle 52 cards.
3. Go back to main menu and choose another option.

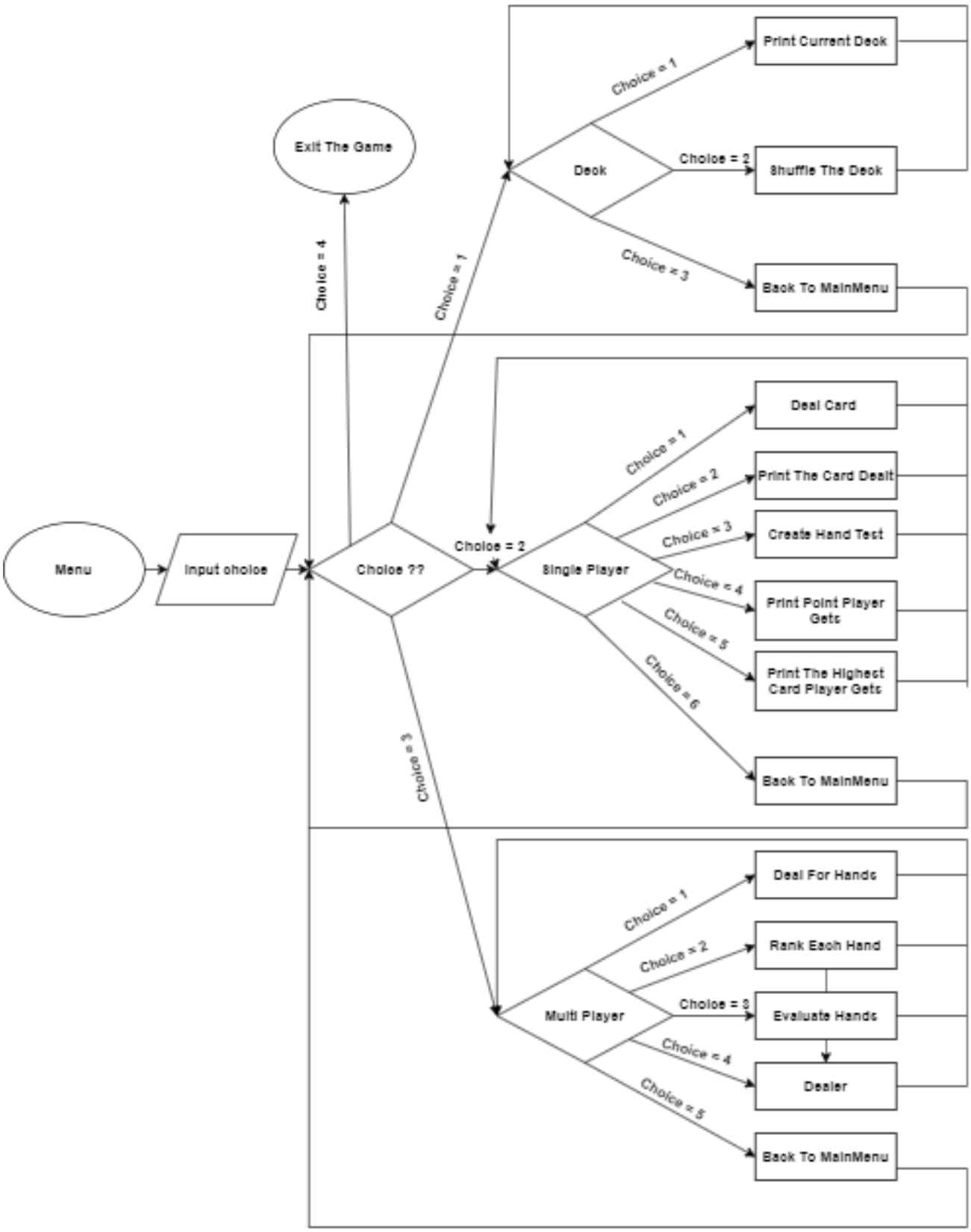
5. void chooseSingleplayer(int deck[SUITS][FACES],int hand);**

- In this option, we have 3 choices :
1. Deal and distribute 5 card for one player.
2. Print 5 cards.
3. Search in print hand function and pick up card from deck to check whether 5 cards has Four of a kind, Full house, Three of a kind,etc…

6. void chooseMultiplayer(int deck[SUITS][FACES], int* hands, int n);**

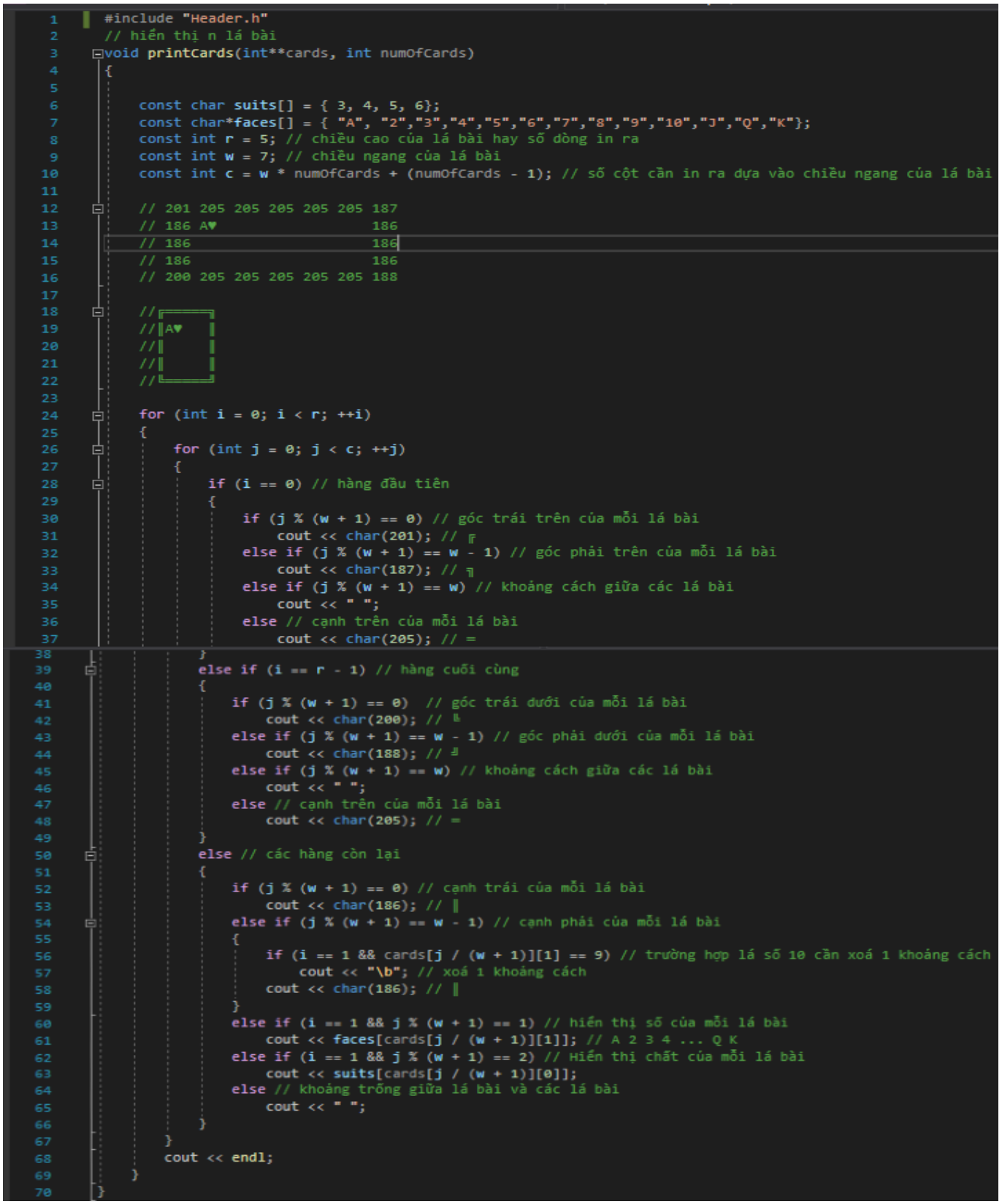
- Variable options is shown. At first you will need to input n of players so the function doesn't get wrong during running (to reset n, back and join Multiplayer again). Dealing for hands is a must, same happened case in Singleplayer to do every further function. Ranking Hands function shows the rank of player in one match, while Evaluate Hands function shows the result of s match and congratulates the highest one at the end. The dealer function moves us to the question 3, where Dealer's side is an NPC, it required us
a huge effort to imagine out the steps to finish the function.

III. Flowchart



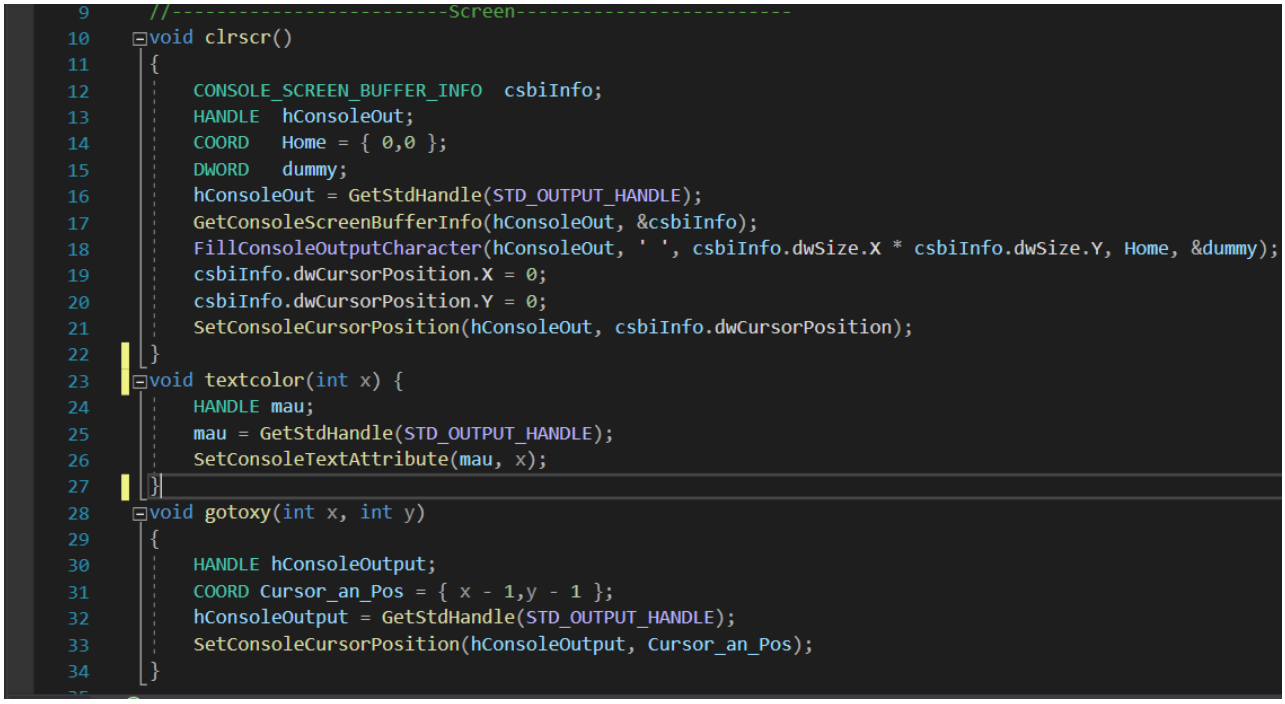
IV. Functions' Extensions

- I will use the ASCII charset to create a card, like this:



```
#include "Header.h"
// hiển thị n lá bài
void printCards(int**cards, int numOfCards)
{

    const char suits[] = { 3, 4, 5, 6};
    const char*faces[] = { "A", "2","3","4","5","6","7","8","9","10","J","Q","K"};
    const int r = 5; // chiều cao của lá bài hay số dòng in ra
    const int w = 7; // chiều ngang của lá bài
    const int c = w * numOfCards + (numOfCards - 1); // số cột cần in ra dựa vào chiều ngang của lá bài

    // 201 205 205 205 205 205 187
    // 186 A♥                  186
    // 186                     186
    // 186                     186
    // 200 205 205 205 205 205 188

    //╔═════╗
    //║A♥   ║
    //║     ║
    //║     ║
    //╚═════╝

    for (int i = 0; i < r; ++i)
    {
        for (int j = 0; j < c; ++j)
        {
            if (i == 0) // hàng đầu tiên
            {
                if (j % (w + 1) == 0) // góc trái trên của mỗi lá bài
                    cout << char(201); // ╔
                else if (j % (w + 1) == w - 1) // góc phải trên của mỗi lá bài
                    cout << char(187); // ╗
                else if (j % (w + 1) == w) // khoảng cách giữa các lá bài
                    cout << " ";
                else // cạnh trên của mỗi lá bài
                    cout << char(205); // ═
            }
            else if (i == r - 1) // hàng cuối cùng
            {
                if (j % (w + 1) == 0)  // góc trái dưới của mỗi lá bài
                    cout << char(200); // ╚
                else if (j % (w + 1) == w - 1) // góc phải dưới của mỗi lá bài
                    cout << char(188); // ╝
                else if (j % (w + 1) == w) // khoảng cách giữa các lá bài
                    cout << " ";
                else // cạnh trên của mỗi lá bài
                    cout << char(205); // ═
            }
            else // các hàng còn lại
            {
                if (j % (w + 1) == 0) // cạnh trái của mỗi lá bài
                    cout << char(186); // ║
                else if (j % (w + 1) == w - 1) // cạnh phải của mỗi lá bài
                {
                    if (i == 1 && cards[j / (w + 1)][1] == 9) // trường hợp lá số 10 cần xoá 1 khoảng cách
                        cout << "\b"; // xoá 1 khoảng cách
                    cout << char(186); // ║
                }
                else if (i == 1 && j % (w + 1) == 1) // hiển thị số của mỗi lá bài
                    cout << faces[cards[j / (w + 1)][1]]; // A 2 3 4 ... Q K
                else if (i == 1 && j % (w + 1) == 2) // Hiển thị chất của mỗi lá bài
                    cout << suits[cards[j / (w + 1)][0]];
                else // khoảng trống giữa lá bài và các lá bài
                    cout << " ";
            }
        }
        cout << endl;
    }
}
```

- We use these functions to make the console more colorful and eye-catching:



```
//-------------------------Screen-------------------------
void clrscr()
{
    CONSOLE_SCREEN_BUFFER_INFO  csbiInfo;
    HANDLE  hConsoleOut;
    COORD   Home = { 0,0 };
    DWORD   dummy;
    hConsoleOut = GetStdHandle(STD_OUTPUT_HANDLE);
    GetConsoleScreenBufferInfo(hConsoleOut, &csbiInfo);
    FillConsoleOutputCharacter(hConsoleOut, ' ', csbiInfo.dwSize.X * csbiInfo.dwSize.Y, Home, &dummy);
    csbiInfo.dwCursorPosition.X = 0;
    csbiInfo.dwCursorPosition.Y = 0;
    SetConsoleCursorPosition(hConsoleOut, csbiInfo.dwCursorPosition);
}
void textcolor(int x) {
    HANDLE mau;
    mau = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(mau, x);
}
void gotoxy(int x, int y)
{
    HANDLE hConsoleOutput;
    COORD Cursor_an_Pos = { x - 1,y - 1 };
    hConsoleOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleCursorPosition(hConsoleOutput, Cursor_an_Pos);
}
```

V. References

1.https://codeshare.io/5gZk8D?fbclid=IwAR1SqihTE2V0qP9wFBJliWcajuwbduD9KKoquBPHZ_64yIid8J_HwFh28W4

2.http://www.cplusplus.com/?fbclid=IwAR3TEVQi3bwUqueW5FnLaPeqIQvEJpmPJ2pF9GgxIabyP2d1pqwuana1RU

3.http://diendan.congdongcviet.com/threads/t353832::chon-ngau-nhienkhongtrunglap.cpp?fbclid=IwAR2mQCml2Yq8YyzMhJJjmjNpt4as5amLWmmjMEWAg5FUejuPQM8W66Wqnw