REPORT
MIDTERM PROJECT 19CLC6

19127622_Ngô Trường Tuyển
19127381_Trần Đức Duy

TABLE OF CONTENTS

I. GROUP MEMBER:

Student_ID	Full name
19127381	Trần Đức Duy
19127622	Ngô Trường Tuyển

II. WORK PROGRESS AND ASSIGNMENT:

1. **Saturday, 18th Apr 2020:**
   - We study the project content and then assign the work.
   - We write code together in part I.1 and sentences a, b, l of part I.2.1.
   - Duy write code for the sentences e, g, i, k in part I.2.1.
   - Tuyen write code for the sentences d, f, h, j in part I.2.1
2. **Sunday, 19th Apr 2020:**
   - We check our code and fix bugs in part I.2.1.
   - We write code together for sentences a, b of part I.2.2.
3. **Wednesday, 22th Apr 2020:**
   - Today, teaching assistant explain the places where students are not still understant in the project. Therefore, we check our code.
   - We write code together for sentences c, d of part I.2.2.
4. **Thursday, 23th Apr 2020:**
   - We design the menu of poker game and sort our code.
   - We separate poker game file into .cpp, .h files.
   - We fix bugs and write additional code for part I.2.1 and I.2.2
5. **Sunday, 26th Apr 2020:**
   - We write the project report.
   - We write code the part I.2.3.
   - We improve our code to make it better.
6. **Tuesday, 28th Apr 2020:**

- We finish project report.
- We finish the part I.2.3 and think idea for part I.2.4.

III. SOLVING THE FUNCTIONS:

- **The rules of the poker game.**
  The Ace card is the lowest card and the King card is the highest card.
- **New functions and name variables have been added:**
  ***Note:** Becauese when We use `const char* suits[SUITS]` and `const char* ranks[RANKS]` instead of `char* suits[SUITS]` and `char* ranks[RANKS]`.

```
const int TOTALCARDS = 52;
```

The sum of cards in a deck.

```
const int SUITS = 4;
```

The suits in a deck.

```
const int RANKS = 13;
```

The ranks in a deck.

```
const int NUMCARDS = 5;
```

The sum of cards in the single player's hand

```
const int NUMDESCRIP = 2;
```

The descriptions in the one card.

```
- isMenu():
```

It is the menu of poker game.

```
- bool checkValueOfShuffleCard(intvalueArray[], int elements, int number)
```

This function will be check the value of number variable and the each the elements of the valueArray array. Return 0 if, the value of number variable equal to one of the value of the valueArray array and otherwise return 1.

```
- void sortByBubbleSort(int** hand)
```

This funtion will be arranged the cards in the player's hand in the ascending order of rank. It is convenient for us to checking hand-ranking categories.

```
- bool isSequentialRank(int** hand)
```

This function will be check the cards in the player's hand, which are the cards of sequential rank or are not the cards of sequential rank. Return true if they are the cards of sequential rank and otherwise return false.

```
- int** playingCard(int** hand)
```

In this function, we initialize a 2-D array 52 by 2, the row is the cards, the column is the description of one card. We use it to store the position row and position column of the card in `deck` array.

```
- int* sumOfS(int** hand)
```

In this function, we initialize a 1-D array to store point of each person(contain machine) in 1 turn. And finally, we return this array.

- **The required functions:**

  - Initialize a `deck` matrix with 4 by 13 and assign the value is 0.

  - The rows is the card's suits including "Hearts", "Diamonds", "Clubs", "Spades".

  - The columns is the card's ranks including "Ace", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King".

  ```
  void shuffleCards(int deck[][])
  ```

  In this function, we will randomly take the values and then assign to the `deck` array. We initialize a 1-D array to store this values, and we need call `checkValueOfShuffleCard` function to check the same values. If it is the same value, we will randomly take the new value and replace it.

  ```
  void printCardsShuffling(int deck[][], const char* suits[],
  const char* ranks[])
  ```

  In this function, we will find th value of card in `deck` array and print out.

  1. **Single Player:**

     a) `int** dealingForHand(int deck[SUITS][FACES])`

  This function allows the player to choose 2 options, case 1 is to take any 5 cards, case two is to take 5 cards in the order the player requests.

  In case one: the machine takes any 5 numbers into a one-dimensional array, then we will look in the deck with the same number assigned to the level 2 pointer, in the first column stored the address of the line and column The second stored the column address to return the result.

  In case 2: the user enters any 5 numbers from 1-52 and then we go into a one-dimensional array, then we find the card with that number in the 2nd level pointer, in the first column stored the address of the line, The second column stored the column address and returns the result.

     b) `void printHand(int** hand, const char* suits[], const char* ranks[])`

  At the level-2 hand pointer, there are 5 cards corresponding to 5 row of the hand array, each row has 2 columns, the first column stores the row position of suit in deck matrix, and the second column stores the column position of rank in deck matrix. And finally, we will use that positions to access the deck matrix and print out the player's cards.

     c) `int** createHandTest(int deck[SUITS][FACES]), int a[])`

  In this function, we will pass 1-D array, it contains the value of cards. We initialize 2-D array to store the row, column postion of one card when we find it out in deck matrix. And finally, we return this 2-D array.

     d) `int isFourOfAKind(int** hand)`

  In this function, we will return 1 if the four cards of one rank, one card of another rank and otherwise return 0.

Therefore, we use nest two for loops to compare the one card and the remaining cards, and use counter variable to count with the start value is 1. If the value of counter variable equal to 4, return 1. And of course, otherwise return 0.

e) `int isFullHouse(int** hand)`

In function, we will return 1 if the three cards of one rank, two cards of another rank and otherwise return 0.

The solution is to use a sequential search algorithm, count the number of duplicates, if the count > 3 then return 0, if the count = 3 then continue to check with the other 2 numbers with the same way as above if the count = 2 then returns 1. If the count != 2 then returns 0.

f) `int isFlush(int** hand)`

In function, we will return 1 if the five cards all of the same suit, not all of sequential rank and otherwise return 0.

Therefore, if the first card and the remaining four cards are different in suit, return 0. If the cards are consecutive rank, return 0, in here, we use the isAscending function to check. And of course, if it passes the above two tests, return 1.

g) `int isStraight(int** hand)`

In function, we will return 1 if the five cards of sequential rank, not all of the same suit and otherwise return 0.

First check if it is consecutive or not by comparing whether a[i] + 1 = a[i + 1] or not, if there exists an unsatisfactory pair then returns 0. Then check to see if the suit is different. Just a different pair exists, returns 1. If all 5 cards are checked without any different suits, then returns 0.

h) `int isStraightFlush(int** hand)`

In function, we will return 1 if the five cards of sequential rank, all of the same suit and otherwise return 0.

Therefore, if the first card and the remaining four cards are different in suit, return 0. If the cards are not consecutive rank, return 0, in here, we use the isAscending function to check. And of course, if it passes the above two tests, return 1.

i) `int isThreeOfAKind(int** hand)`

This function requires checking to see if these 5 cards have 3 cards of the same rank and 2 other cards of the same rank.

The solution is to use a sequential search algorithm, count the number of duplicates, if the count > 3 then return 0, if the count = 3 then continue to check with the other 2 numbers with the same way as above if the count = 2 then returns 0. If the count != 2 then returns 1.

j) `int isTwoPairs(int** hand)`

In function, we will return 1 if two cards of on rank, two cards of another rank, one card of a third rank and otherwise return 0.

We use nest two for loop. The outer loop will loop through each card, inner loop will loop from the card of the outer loop plus 1 until the last card and if they have

the same rank, counter variable will be increasing by 1. The loops finish, we check the counter variable, return 1 if counter variable equal 2 and otherwise return 0.

k) `int isPair(int** hand)`

In function, we will return 1 if two cards of one rank, three cards of three other rank and otherwise return 0.

We use nest two for loop. The outer loop will loop through each card, inner loop will loop from the card of the outer loop plus 1 until the last card and if they have the same rank, counter variable will be increasing by 1. The loops finish, we check the counter variable, return 1 if counter variable equal 1 and otherwise return 0.

l) `int getHighestCard(int** hand)`

According to the rules set out in IV. SOLVING THE FUNCTION, the ace is the lowest card and the king is the highest. And after finshing the dealing for hand, we will sort the cards in the player's hand in ascending order. Therefore, the highest card is the last card in hand matrix.

2. Multiple Player:

a) `int*** dealingForHands(int deck[SUITS][FACES], int n)`

The user will choose the number of players, assign to n variable.

We will use 3-D array by the technique of allocating memory. The first level is n players, the second level is 5 cards and the last level is the description about that card, which contains suit and rank.

We write new function, its aim is return a level-2 pointer with 52 rows, it run from 1 to 52.

After we have the deck of card, we arranged sequentially. Then we dealing for each player by the following way: we will deal the 5 cards for the i^{th} player in order the next card will be bigger than the previous card by n units(n is the number of players). For example, if there are 4 player, the first player will take the following card {1, 5, 9, 13, 17} and so on. And finally, we will copy the that cards to the 3-level pointer.

b) `int getStatusOfHand(int** hand)`

In this function, we must to get the highest score of the cards in the player's hand. Therefore, we will check that cards with the functions in the hand-ranking categories from best to worst. If that cards is true in a certain case(in order from best to worst), we return the score of this case and break the checking with that cards.

c) `int* rankingHands(int*** hands, int n)`

In this function, we will use level-2 pointer to stores one player's cards and then we calculate each player's score and assign in a level-1 pointer to store point, and creat new level-1 pointer to store the player's position corresponding. After we arrange in descending the points and position corresponding, with the highest score at a[1]. Return level-1 pointer points to the array stores this positions.

d) `int* evaluateHands(int deck[SUITS][RANKS], int n, int s)`

For each times of dealing cards, we will shuffling and dealing cards again, then we calculating points by sumOfS function (similar to the rankingHands function but sumOfS function returning the player's score). sumOfS function returns a level-1 pointer that stores points of n players in 1 turn. The i^{th} rank is located in a[i] of a array. And we need creat a level-1 pointer,which stores the total score of s times dealing cards. And finally, we return this array.

3. Man vs Machine:

```
void machineVsPlayer(int deck[SUITS][RANKS], const char* suits[], const char* ranks[])
```

In this function, we have a function play card between the dealer and other players. We use dealingForHands function for n players and the dealer is the first peopler. The dealer will choose two options: The first, it is random replacement and the second, it is replace to get better situation.
In case one: the dealer will be exchanged for maximum of three cards. When exchanging, the dealer may select one of the five available cards, and receive one of the remaining card in the deck.
In case two: the dealer will see the remaining card in deck of cards and exchange card to get better, it is similar is case one.

4. The dealer:

```
void isDealer(int deck[SUITS][RANKS], const char* suits[], const char* ranks[])
```

In this function, we have a function play card between the dealer and one player. Use the playingCard function to store the deck. Both dealer and player can choose to draw(max is 3 cards) or not. If they want to exchange, they will exchange the card that them wants to exchange with the card they want to receive from the remaining cards. The remaining cards start from (2 * 5) + 1.

5. The mode:

- Easy: Can exchange 3 cards.

- Medium: Can exchange 2 cards.

- Hard: Can exchange 1 cards

IV. REFERRENCES:

We do not references.