

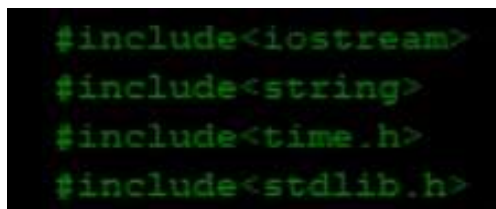
Single-Player Poker Game Report

Using OOP concepts in C++, this lab compiles a single player Poker game with a Base Class and Class Methods. The user starts with 100 chips (\$1.00 each) and is given a randomized hand of four cards from a deck of 36 cards. Winning possibilities include 4ofAKind, Straight Flush, Flush, Straight, 3ofAKind, Two Pair, Pair, and Bubkiss. This can be played as long as the user wishes to play and has enough chips. A Final report will be generated with amount of wins, losses, hands, and net winnings.

Game Description:

1. Player starts with 100 chips, each worth \$1.00
2. Deck of 36 cards: each with a number (1-9) and a suit (Clubs, Spades, Hearts, Diamonds)
3. Before each hand is dealt, the deck is shuffled, and the user is asked if they want to place a bet ranging from 1-#of chips and then will be given the top 4 cards of the deck
4. The program will then test the hand amongst various conditions to see if there is any winning possibility with any of the following ways:
 - a. 4 of a kind - 400 chips for each chip that was placed
 - b. Straight Flush - 300 chips for each chip that was placed
 - c. Flush - 250 chips for each chip that was placed
 - d. Straight - 200 chips for each chip that was placed
 - e. 3 of a kind - 150 chips for each chip that was placed
 - f. Two Pair - 100 chips for each chip that was placed
 - g. Pair - 1 chip for each chip that was placed
 - h. Bubkiss - Loses their bet
5. The program will run as long as the user would like to play or until they run out of chips
6. At the end, a final report with a summary of the game will be generated with the following components:
 - a. # of Hands the user played
 - b. # of Bets won
 - c. # of Bets lost
 - d. Net Winnings (Final - 100)

In this Final Report, I will be discussing each major part of my code and explain my thought process along with the technicalities behind it. I have used a variety of techniques like Classes, Methods, Arrays, If-Else Statements, Functions, etc. to re-create Poker in C++. I will be sharing screenshots of my program and explaining my thought process and technicalities behind each section of my code. Please refer to the provided figures above descriptions for reference.



```
#include<iostream>
#include<string>
#include<time.h>
#include<stdlib.h>
```

Figure 1

Figure 1: This is a screenshot of all the libraries I have decided to include. The libraries `iostream` and `string` allow for input/output and string formatting respectively. The header files `time` and `stdlib` allow for `srand` functions and memory allocation services respectively.

```
using namespace std;
//Poker Superclass with all methods and arrays
class Poker
{
public:
    int num [9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    char suit[4] ={'C', 'S', 'H', 'D'};
    int n[4];
    char s[4];
    int bank;
    int bet;
    int count;
    int wins;
    int losses;
    int net;

    Poker();
    void displayHand();
    int checkWin();
};
```

Figure 2

Figure 2: This screenshot contains the Poker Superclass and all the methods and functions that are contained within. I have made all the methods, functions, and variables public in this scenario to make things easier and more accessible throughout the program. In the superclass, I have initialized the two arrays that contain the number ranges and suits and two empty arrays that would be used for the shuffling of the deck. In addition, I created a displayHand function that would generate a random hand every time and call the checkWin function that would check its possibilities. The constructor is also called to initialize all the above variables.

Figure 3: Displays the main function that's used to create an object of the Poker class and then is used to call the displayHand function of the superclass.

```
int main(){
    Poker poker;
    poker.displayHand();
}
```

Figure 3

Figure 4: Displays the constructor Poker() that was used to initialize the variables declared in the superclass

```
Poker :: Poker()
{
    bank = 100;
    bet = 0;
    count = 0;
    wins = 0;
    losses = 0;
    net = 0;
}
```

Figure 4

```
// Function that displays the hand, every win/loss, and generates the final report
void Poker :: displayHand()
{
    Poker poker;
    char inp;
    int stop = 0;
    cout<<"Welcome to single player Poker game!"<<endl;
    cout<<"Your initial bank roll is $100.00"<<endl;

    do
    {
        cout<<"-----"<<endl;
        cout <<"Play a hand [y/n]?"<<endl;
        cin >> inp;
        if (bank == 0){
            cout <<"You have used up all your chips. Sorry!"<<endl;
            inp = 'n';
        }
        if (inp == 'y'){
            count++;
            cout<<"Place your bet[1, "<<bank<<"]:"<<endl;
            cin >> bet;
            if (bet <= bank && bet >= 1){
                cout << "...Shuffling Deck..."<<endl;
                cout << "Let the cards fall where they may..."<<endl;
                srand(time(NULL));
                for (int i = 0; i < 4; i++){
                    n[i]=num[rand()%9];
                    s[i]=suit[rand()%4];
                    cout << n[i] << s[i] << " ";
                }
                checkWin();
                stop = 0;
            }
        }
    }
}
```

Figure 5

Figure 5: This part of the code resembles the highlight of the code where all the processing takes place. It contains the initial welcome statements and prompts user to place a bet or not. Based off the user's response, the function will then go into a series of if-else statements under a big do-while loop so that the user can play as long as they want to or until they have run out of chips. It registers that user's input as a character and then uses if-else statements to determine what action to take. In this case, if the user says 'y', the program will then go on to shuffle a hand with the for loop and the empty arrays declared from the superclass. After the hand has been generated, it then calls the checkWin function which checks if they won.

```

// Checks for different winning possibilities
int Poker :: checkWin()
{
    //4ofAKind
    if (n[0]==n[1] && n[0] == n[2] && n[0] == n[3])
    {
        wins++;
        bank += (400 * bet);
        cout <<endl<< "Congrats: You got a 4ofAKind and have won $400 for eve
    }
    // Flush
    else if (s[0] == s[1] && s[0] == s[2] && s[0] == s[3])
    {
        wins++;
        int temp;
        int temp2;
        for (int t = 0; t<3; t++){
            temp = t;
            for (int j = t+1; j<4; j++){
                if (n[j]<n[temp])
                {
                    temp=j;
                }
            }
            temp2 = n[t];
            n[t]=n[temp];
            n[temp]=temp2;
        }
        // Straight Flush
        if (1+n[0]==n[1] && 1+n[1] == n[2] && 1+n[2] == n[3])
        {
            bank += (300 * bet);
            cout << endl<<"Congrats: You got a Straight Flush and have won $3
    }
}

```

Figure 6

Figure 6: This portion of the code is the brains of the code which looks for the multiple winning possibilities: 4ofAKind, Straight Flush, Flush, Straight, 3ofAKind, Two Pair, Pair, or Bubkiss. This is also another set of multiple if-else statements that compare the suits array and the number arrays separately and then decide what the user has won, or if they even won anything. Within a lot of the if-else statements, there are several techniques used for comparing the arrays. Mainly, I have used or (||) & and (&&) comparisons with arrays and their indexes to compare. For the wins like Straight and Straight Flush, I used nested for-loops in order to re-arrange the numbers and figure out if they form a consecutive order. After running through the if-else statements, the program will decide one (or more, if there are multiple possibilities) type of win and adds the respective amount times the bet and adds it to the existing bank.

```

    else{
        count--;
        cout<<"Invalid Amount. Please try again."<<endl;
    }
}
else if (inp == 'n'){
    net = bank - 100;
    cout<<"Thanks for playing..."<<endl;
    cout<<"You played a total of "<<count<<" hands"<<endl;
    cout<<"Of which you won "<<wins<<"<<endl;
    cout<<"And you lost "<<losses<<"<<endl;
    cout<<"But in the end you won $"<<net<<"<<endl;
    stop = 1;
}
else{
    cout<<"Invalid Choice. Please choose again."<< endl;
}
}
while (stop == 0);

```

Figure 7

Figure 7: This includes the second half of Figure 5 that has the if-else condition if the user decides to stop playing or has ran out of chips. In the end of the program, a final report will be generated that contains a summary of the user's game that contains the number of hands played, wins, losses, and net winnings. The program also tests against invalid inputs and prompts the user to enter their desired choice once again. The final summary is determined after the functions have used the ++ operator and increased every time the user had played.

SAMPLE OUTPUTS

As you can see in the sample output below, the program plays a single-player Poker game that tests against various winning possibilities and goes on until the user is willing to play or has run out of chips. The final report is also generated with a summary of the game at the end. Getting every possibility to appear is difficult as the program randomizes the deck each time.

```
Welcome to single player Poker game!
Your initial bank roll is $100.00
-----
Play a hand [y/n]?
y
Place your bet[1, 100]:
10
...Shuffling Deck...
Let the cards fall where they may...
2D 2C 6S 5S
Congrats: You got a Pair and have won $1 for every chip you placed!
-----
Play a hand [y/n]?
y
Place your bet[1, 110]:
10
...Shuffling Deck...
Let the cards fall where they may...
4S 9C 2S 5S
Sorry: you got Bubkiss and have lost $10
-----
Play a hand [y/n]?
y
Place your bet[1, 100]:
110
Invalid Amount. Please try again.
-----
Play a hand [y/n]?
y
Place your bet[1, 100]:
10
...Shuffling Deck...
Let the cards fall where they may...
4H 6C 4D 7S
Congrats: You got a Pair and have won $1 for every chip you placed!
-----
Play a hand [y/n]?
n
Thanks for playing...
You played a total of 3 hands
Of which you won 2
And you lost 1
But in the end you won $10
-----
Process exited with return value 0
Press any key to continue . . .
```