**Alejandro Martinez**

**Programming 3**

**Program #4 Poker Stud**

**Pseudocode**

**Display program purpose:**

This program simulates a game of poker. All the hands are displayed, then they are shuffled, then the are given to the players, then each player sorts each hand, then each hand is classified according to ranks given in poker stars, then we find a winner. Lastly we have pre-determined hands that will be reviewed and classified their poker rank output.

The input as described above should be within the boundaries described.

After the input is validated, the program will generate a deck of cards structs. This deck of cards structs is generated from (1) to (53). The reason why the deck array did not started at 0 was to facilitate further computations.

After the array is generated, internal manipulation was done to order the deck. Then the modern algorithm of shuffling documented below was used to shuffle the deck of card structs.

Then a struct of hands was created to store the hands of each player. Then these structs of hands were dealt. Then they were sorted using the algorithm of insertion sort.

Then using poker starts we classified each hand. With this process we were able to select a winner.

Lastly, we used pre-determined hands and classified them with the same algorithms previously described.

**Display results:** Using int validateInput (int argc, char \*argv[]); int checkArgumentCount(int argc); int checkInputLength(char \*argv[]); int checkStringToInt(char \*argv[]); int checkNumberRange(char \*argv[]); void messageToPrintForError (); void explainTheLegend(const char SUITSARRAY[SUITS], const char RANKARRAY[FACES]);

void populateDeckOfCards(Card deck[]); int whatCard (int cardInTheDeck);

int whatSuit (int cardInDeck); void printDeckOfCards (Card mdeck[], const char SUITSARRAY[], const char RANKARRAY[]); void shuffle (Card deck []);

void swap (Card \*currentPosition, Card \*newPosition); void createTheHands(Hand handPerPlayer[], Card deck[CARDDECKSIZE], const int playersInTheGame);

void printTheHands(Hand handPerPlayer[],const char SUITSARRAY[SUITS], const char RANKARRAY[FACES],const int numberOfPlayers); void insertionSort(Hand handPerPlayer[], const int playersToSort); int highCard(Hand handProvided); int isItOnePair (Hand handProvided); int isItTwoPair (Hand handProvided); int isItThreeOfAKind (Hand handProvided); int isItAStraight (Hand handProvided); int isItAFlush (Hand handProvided);

int isItAFullHouse (Hand handProvided); int isItFourOfAKind (Hand handProvided);

int straightFlush(Hand handProvided); void printResults(Hand handPerPlayer[],const char SUITSARRAY[SUITS], const char RANKARRAY[FACES],const int numberOfPlayers);

void classificationOfHands (Hand \*handsAtTable); void printWinner(Hand handPerPlayer[],const char SUITSARRAY[SUITS], const char RANKARRAY[FACES],const int numberOfPlayers); int determineAWinner (Hand handPerPlayer[], const int numberOfPlayers); void printRankMessage(Hand handsAtTable); void assignRanks (Hand playersInTheGame[], const int players); void pokerTest (Hand pokerHands[]);

void printTestingHands(Hand pokerHands[], const char SUITSARRAY[SUITS], const char RANKARRAY[FACES]);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Source File: Main.c

The source file Main.c will call methods from the source files inputValidation.c and poker.c. It will also use the header file cardConstants.h to access constants, enums and structs.

The main() in the Main.c will be the only method available and will run the methods required to print what is in the instructions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Source File: inputValidation will hold all the methods to validate the input required. The objective of this source file to validate input that is not a string, or numbers that are not in the range from 1-13 or 2-10 as stated in the pre-requisites. Also please recall that the first input will always be changed to 5.

The set of functions required are as follows:

int validateInput (int argc, char \*argv[])

* This function will be used as the brain of the validating input
* Call functions checkArgumentCount(argc) &&checkInputLength(argv) &&checkStringToInt (argv)&&checkNumberRange(argv))
* If all these functions are true then return true. Else return False.

int checkArgumentCount(int argc)

* Validates the amount of inputs

int checkInputLength (char \*argv[])

* Checks the length of all inputs with strlen
* If the inpurs happened to have a length below 2 characters then we are good

int checkStringToInt (char \*argv[])

* Checks if arguments can be converted to digits.
* If this is the case return true

int checkNumberRange(char \*argv[])

* This checks the number range.
* For input #1 range must be from [1-13] and the input will change to 5
* For input #2 range must be from [2-10]

void messageToPrintForError ()

* If none of the statements are true the program exits
* The program will displayed the stated error.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Source File poker.c will hold the functions for creating the deck of card structs, printing this deck of card structs, dealing hands, shuffling the hands, classifying the hands, determining winners, analyzing pre-determined hands.

void explainTheLegend(const char SUITSARRAY[SUITS], const char RANKARRAY[FACES])

-Print Detailed explanation of the type of suits and ranks we have

void populateDeckOfCards(Card deck[CARDDECKSIZE])

* Populate the deck array with the Card structs.
* The Structs will hold the rank and suit value

void printDeckOfCards (Card mdeck[], const char SUITSARRAY[SUITS], const char RANKARRAY[FACES])

* Started the array at index COUNTER\_1 (1), as it in later functions it will be easier to manipulate to print the order array of structs
* Recall that the struct of cards he have has a rank and a suit.
* Print the whole array of Card Structs

int whatCard (int cardInTheDeck)

* Manipulate the number as follows
* indexOfCardArray = cardInTheDeck % FACES;
* if (indexOfCardArray == 0)

actualIndexInFaceArray = FACES - COUNTER\_ONE;

else

actualIndexInFaceArray = indexOfCardArray - COUNTER\_ONE;

int whatSuit (int cardInDeck)

* This function will return the index of what suit to pick in the suit array.
* Using ifs statements for bounds from 1-53 to know what type of suit.

void shuffle (Card deck [])

* Here we used the srandom(time(NULL))defined in main

to generate a seed that really is random, as it is attached to

time.

* We loop through our original array of cards and we

shuffle the array with this seed generated using the

formula:

randomLocation = COUNTER\_ONE + (rand() % CARDS);

void swap (Card \*currentPosition, Card \*newPosition)

* Swap pointers

void createTheHands (Hand handPerPlayer[], Card deck[CARDDECKSIZE],

const int playersInTheGame)

- Loop through the total number of players

- In each player create the hand

- The cards come from the shuffle array

void printTheHands(Hand handPerPlayer[],const char SUITSARRAY[SUITS],

const char RANKARRAY[FACES],const int numberOfPlayers)

- Loop through the total number of players

- Inside Loop of total number of cards

void insertionSort(Hand handPerPlayer[], const int playersToSort)

- Use constant arrays to print the actual cards

- Get a list of unsorted numbers

- Set a marker for the sorted section after

the first number in the list

- Repeat steps 4 through 6 until the unsorted

section is empty.

- Select the first unsorted number.

- Swap this number to the left until it arrives at

the correct sorted position.

- Advance the marker to the right one position

- Stop

int highCard(Hand handProvided)

* Return true if we are ever in this condition

int isItOnePair (Hand handProvided)

- Method will receive the hand to classify

- Assign the rank value of each card to an int

- As we happened to have sorted cards the following

algorithm will apply:

\* If CardOne is equal to CardTwo and the rest of the

the cards are not the same then we have a pair.

\* If CardTwo is equal to CardThree and the rest of the

the cards are not the same then we have a pair.

\* If CardThree is equal to CardFour and the rest of the

the cards are not the same then we have a pair.

\* If CardFour is equal to CardFive and the rest of the

the cards are not the same then we have a pair.

- If none of these conditions are met then we do not have a

a pair.

int isItTwoPair (Hand handProvided)

- Method will receive the hand to classify.

- Assign the rank value of each card to an int.

- As we happened to have sorted cards the following.

algorithm will apply:

\* If CardOne is equal to CardTwo, and CardThree is

equal to CardFour and CardFive is not equal to

CardFour and CardThree is not equal to CardTwo,

we have two distinct pairs.

\* If CardTwo is equal to CardThree, and CardFour is

equal to CardFive and CardOne is not equal to

CardTwo and CardThree is not equal to CardFour,

we have two distinct pairs.

- If none of these conditions are met then we do not have a

two pairs.

int isItThreeOfAKind (Hand handProvided)

- Method will receive the hand to classify.

- Assign the rank value of each card to an int.

- As we happened to have sorted cards the following.

algorithm will apply:

\* If CardOne is equal to CardTwo, and CardTwo is

equal to CardThree and CardThree is not equal to

CardFour and CardFour is not equal to CardFive,

we have three of a kind.

\* If CardTwo is equal to CardThree, and CardThree is

equal to CardFour and CardOne is not equal to

CardTwo and CardFour is not equal to CardFive,

we have three of a kind.

\* If CardThree is equal to CardFour, and CardFour is

equal to CardFive and CardOne is not equal to

CardTwo and CardTwo is not equal to CardThree,

we have three of a kind.

- If none of this conditions are met then we do not have a

three cards of the same kind.

int isItAStraight (Hand handProvided)

- Method will receive the hand to classify.

- Assign the rank value of each card to an int.

- As we happened to have sorted cards the following.

algorithm will apply:

\* If CardTwo is equal to CardOne plus one,

and CardThree is equal to CardTwo plus one,

and CardFour is equal to CardThree plus one,

and CardFive is equal to CardFour plus one,

return True, and we have a straight.

- If none of these conditions are met then we do not have a

a straight.

int isItAFlush (Hand handProvided)

- Method will receive the hand to classify.

- Assign the suit value of each card to an int.

- As we happened to have sorted cards the following.

algorithm will apply:

\* If CardOne is equal to CardTwo,

and CardTwo is equal to CardThree,

and CardThree is equal to CardFour,

and CardFour is equal to CardFive,

return True, and we have a flush.

- If none of these conditions are met then we do not have a

a flush.

int isItAFullHouse (Hand handProvided)

- Method will receive the hand to classify.

- If the methods isItOnePair(handProvided) &&

isItThreeOfAKind(handProvided) return True then we

have a full house.

- If none of this conditions are met then we do not have a

a full house.

int isItFourOfAKind (Hand handProvided)

- Method will receive the hand to classify.

- Assign the rank value of each card to an int.

- As we happened to have sorted cards the following.

algorithm will apply:

\* If CardOne is equal to CardTwo, and CardTwo is

equal to CardThree and CardThree is equal to

CardFour and CardFour is not equal to CardFive,

we have four of a kind.

\* If CardTwo is equal to CardThree, and CardThree is

equal to CardFour and CardFour is equal to

CardFive and CardOne is not equal to CardTwo,

we have four of a kind.

- If none of this conditions are met then we do not have a

four cards of the same kind.

IMPORTANT NOTE: These cards can be compared as the

algorithm above explains due that they are sorted.

int straightFlush(Hand handProvided)

- Method will receive the hand to classify.

- If the methods isItAStaright(handProvided) &&

isItAFlush(handProvided) return True then we

have a straight flush.

- If none of this conditions are met then we do not have a

a straight flush.

void printResults(Hand handPerPlayer[],const char SUITSARRAY[SUITS],

const char RANKARRAY[FACES],const int numberOfPlayers)

- Loop through the total number of players

- Inside Loop of total number of cards

- Use constant arrays to print the actual cards

- After going through each hand print the result

void classificationOfHands (Hand \*handsAtTable)

- Method will receive the hand to classify through a pointer

- If any of the eight conditions stated above is true

the rankInPoker varibale will recieve a number.

- If the hand does not qualify for any of the eight given

states then we provide the default state high card.

void printWinner(Hand handPerPlayer[],const char SUITSARRAY[SUITS],

const char RANKARRAY[FACES],const int numberOfPlayers)

- Loop through the total number of players

- Inside Loop of total number of cards

- Use constant arrays to print the actual cards

- After going through each hand print the result

- If the player matches the highest rank called in

method determine a winner. The 'winner' will print.

int determineAWinner (Hand handPerPlayer[], const int numberOfPlayers)

- Loop thorough each hand and compare the pokerRank

- As the poker ranks are compared. The one with the

highest number is stored.

void printRankMessage(Hand handsAtTable)

- Assign to an int the poker rank

- Compare it to the 9 different outcomes and state message

void assignRanks (Hand playersInTheGame[], const int players)

- Loop thorough each hand

- Call function classificationOfHands passing a pointer

to each hand.

void pokerTest (Hand pokerHands[])

- This Hands were explicitly typed from the

specifications.

- Assign each hand to each valued of the array.

void printTestingHands(Hand pokerHands[], const char SUITSARRAY[SUITS],

const char RANKARRAY[FACES])

- Loop through the total number of hands

- Assign a poker rank through a classification of

hands function.

- Loop through the total number of hands again

- Loop through each card to print them

- Print the rank number name.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The header file: cardConstants.h will hold the constant and function prototypes shared by the source files of Main.c, poker.c and inputValidation.c

In the structure of our program we will be using to structs described below.

- Struct of Cards will store two integers (Rank, Suit)

- Struct of Hand will store and array of struct cards and an int with the poker rank. (poker rank is a number that describes the type of hand)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_