CIS37A-ClassProject

Due: July-27-2017, 11:59PM

Name: Subal Pant

The point of the Poker Simulation game is to simulate a poker game. The game works in the way that it first deals out cards to two players in an alternating fashion (i.e. the first card goes to the first player and the second card goes to the second player). The players have complete hands when they have five cards each in their hands. They then look at what kind of poker combinations they have in their hand (in order from lowest to highest: pair, double pair, triple, straight, flush, quad; there is high card, full house, straight flush, and royal flush but we don’t cover those in this project). The players then compare hands to see who has the higher poker hand and the player with the higher hand wins. My program solves this problem by assigning individual hands to each player and then checking what kind of poker hand they have. After it compares the poker hands to see which is higher and then declares a winner.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <stdbool.h>

#define CARDS 52

#define FACES 13

#define SUITES 4

#define HAND 5

enum pokerOrder {

noPoker,

singlePair,

doublePair,

triple,

straight,

flush,

quad

};

struct card {

const char \*face;

const char \*suit;

unsigned int nFace;

unsigned int nSuite;

};

typedef struct card Card;

void fillDeck(Card \* const wDeck, const char \* wFace[], const char \* wSuit[]) {

for (size\_t i = 0; i < CARDS; ++i) {

wDeck[i].face = wFace[i % FACES];

wDeck[i].suit = wSuit[i / FACES];

wDeck[i].nFace = i % FACES;

wDeck[i].nSuite = i / FACES;

}

}

void shuffle(Card \* const wDeck) {

for (size\_t i = 0; i < CARDS; ++i) {

int rn = rand();

size\_t j = rn % CARDS;

Card temp = wDeck[i];

wDeck[i] = wDeck[j];

wDeck[j] = temp;

}

}

void deal(const Card \* const wDeck, size\_t size) {

for (size\_t i = 0; i < size; ++i) {

printf("%s of %s\n", wDeck[i].face, wDeck[i].suit);

}

printf("\n");

}

void pokerHand(const Card \* const wDeck, Card \* const wHand, int playerNumber, int numberOfPlayers) {

switch (playerNumber) {

case 0: {

size\_t j = 0;

for (size\_t i = 0; i < HAND; ++i) {

if (numberOfPlayers == 1) {

j = i;

}

else if (numberOfPlayers == 2) {

j = i \* 2;

}

wHand[i] = wDeck[j];

}

break;

}

case 1: {

size\_t j = 0;

for (size\_t i = 0; i < HAND; ++i) {

if (numberOfPlayers == 2) {

j = i \* 2 + 1;

}

wHand[i] = wDeck[j];

}

break;

}

}

}

//Flush Check

enum pokerOrder flushCheck(unsigned int wSuiteCounter[]) {

for (size\_t s = 0; s < SUITES; s++) {

if (wSuiteCounter[s] == 5) {

return flush;

}

}

return noPoker;

}

//Straight Check

enum pokerOrder straightCheck(unsigned int wFaceCounter[]) {

int cn = 0;

for (size\_t i = 0; i < FACES; i++) {

if (cn == 5) {

return straight;

}

if (wFaceCounter[i] == 1) {

cn++;

}

else if (wFaceCounter[i] == 0) {

if (cn > 0) {

cn--;

}

}

}

return noPoker;

}

//Check for Triple, Quad

enum pokerOrder quadTripleCheck(unsigned int wFaceCounter[]) {

for (size\_t s = 0; s < FACES; s++) {

if (wFaceCounter[s] == 4) {

return quad;

}

else if (wFaceCounter[s] == 3) {

return triple;

}

}

return noPoker;

}

//Check for Single or Double Pair

enum pokerOrder doublePairCheck(unsigned int wFaceCounter[]) {

unsigned int pairCounter = 0;

for (size\_t s = 0; s < FACES; s++) {

if (wFaceCounter[s] == 2) {

pairCounter++;

}

}

if (pairCounter == 2) {

return doublePair;

}

if (pairCounter == 1) {

return singlePair;

}

return noPoker;

}

enum pokerOrder checkHand(const Card \* const wHand, size\_t size) {

unsigned int faceCounter[FACES] = { 0 };

unsigned int suiteCounter[SUITES] = { 0 };

for (size\_t i = 0; i < size; i++) {

++faceCounter[wHand[i].nFace];

++suiteCounter[wHand[i].nSuite];

}

enum pokerOrder po;

po = flushCheck(suiteCounter);

if (po != noPoker) {

return po;

}

po = straightCheck(faceCounter);

if (po != noPoker) {

return po;

}

po = quadTripleCheck(faceCounter);

if (po != noPoker) {

return po;

}

po = doublePairCheck(faceCounter);

if (po != noPoker) {

return po;

}

return noPoker;

}

void printPokerOrder(enum pokerOrder po) {

switch (po) {

case noPoker:

printf("No Poker Hand!\n");

break;

case singlePair:

printf("You have a single pair!\n");

break;

case doublePair:

printf("You have a double pair!\n");

break;

case triple:

printf("You have a triple!\n");

break;

case straight:

printf("You have a straight!\n");

break;

case quad:

printf("You have a quadruple!\n");

break;

case flush:

printf("You have a flush!\n");

break;

}

}

int main(void) {

Card deck[CARDS];

Card hand1[HAND];

Card hand2[HAND];

char arr[100][25];

enum pokerOrder pokerCheck1, pokerCheck2;

FILE \*rfPtr;

const char \*face[FACES] = { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };

const char \*suit[SUITES] = { "Hearts", "Diamonds", "Clubs", "Spades" };

srand(time(NULL));

fillDeck(deck, face, suit);

shuffle(deck);

printf("One Player Poker:\n");

pokerHand(deck, hand1, 0, 1);

deal(hand1, HAND);

pokerCheck1 = checkHand(hand1, HAND);

printPokerOrder(pokerCheck1);

for (size\_t i = 0; i < 100; i++) {

fillDeck(deck, face, suit);

shuffle(deck);

printf("\n\n\nTwo Player Poker:\n");

printf("Player 1:\n");

pokerHand(deck, hand1, 0, 2);

deal(hand1, HAND);

pokerCheck1 = checkHand(hand1, HAND);

printPokerOrder(pokerCheck1);

printf("\n");

printf("Player 2:\n");

pokerHand(deck, hand2, 1, 2);

deal(hand2, HAND);

pokerCheck2 = checkHand(hand2, HAND);

printPokerOrder(pokerCheck2);

printf("\n");

if (pokerCheck1 > pokerCheck2) {

printf("Result: Player 1 Won!");

strcpy(arr[i], "Player 1 Won!\n");

}

else if (pokerCheck1 < pokerCheck2) {

printf("Result: Player 2 Won!");

strcpy(arr[i], "Player 2 Won!\n");

}

else {

printf("Result: No contest!");

strcpy(arr[i], "No contest!\n");

}

}

rfPtr = fopen("results.txt", "w");

if (rfPtr == NULL) {

printf("Could not open file.");

return;

}

for (size\_t i = 0; i < 100; i++) {

fprintf(rfPtr, arr[i]);

}

fclose(rfPtr);

getch();

}

The program starts by first defining the sizes of different card sets (52 CARDS, 13 FACES, 4 SUITES, etc.). It then declares an enum that defines the order of poker hands from least to greatest. There is then a struct that declares what values a “card” structure should have. There are then several functions that have different actions. The “fillDeck” function fills a deck of CARDS size with different face/suite cards with no duplicates. The “shuffle” function randomizes the order of the deck. The “deal” function shows the cards that each player drew from the deck in 5 rows. The “pokerHand” function takes the number of players in the game and the player “number” (player 0 is the first player and player 1 is the second player). These parameters are then used to assure that each card is alternated when being assigned to players. The next four functions are the functions that check to see what kind of poker hand the player has (if they don’t have one, it returns noPoker). The “checkHand” function takes a hand and runs it through the previous four functions to figure out what kind of poker hand the player has. The “printPokerOrder” function prints out the result of the poker hand for each player (noPoker prints “No Poker Hand!”). The main function contains all the function calls and the “for” loop that runs the program a hundred times. There is an “if” loop at the bottom of the main function that tells which player won using the return from the enum. The very last function in main stores the results of the hundred games into an array and into a file.

I learned, from this project, that card games are not easy to put into code because there are many, many scenarios that need to be covered in the code. My approach to solving the problem was to cover as many of these scenarios as possible (and I still didn’t cover them all). I used the debugging features of Visual Studios constantly to check the output of my functions and ended up rewriting the pokerHand function so it could incorporate both one and two player games. For the future, I would expand the game so that it includes all poker hands. High card, straight flush, full house, and royal flush were not included in my program. Also a check to compare two of the same poker hand should be implemented so that there are less cases of “No Contest”.