Project 1

Introduction

Early on I was talking with my friends about which project I could do for this class and blackjack was one that seemed reasonable. That in fact was the case, I think it was a reasonable goal to have, a simple game but with hidden complexity on how to implement it I thought. How to simulate a card deck, make an artificial intelligence for the dealer, a betting system simulating the casino experience, maybe even multiple people at table. I tried my best to take notes from you Dr. Lehr and truly iterate with each version of the program, progressing little steps at a time. I am not sure how successful I was at making this “game” but I truly tried my best and had a pretty fun time doing it.

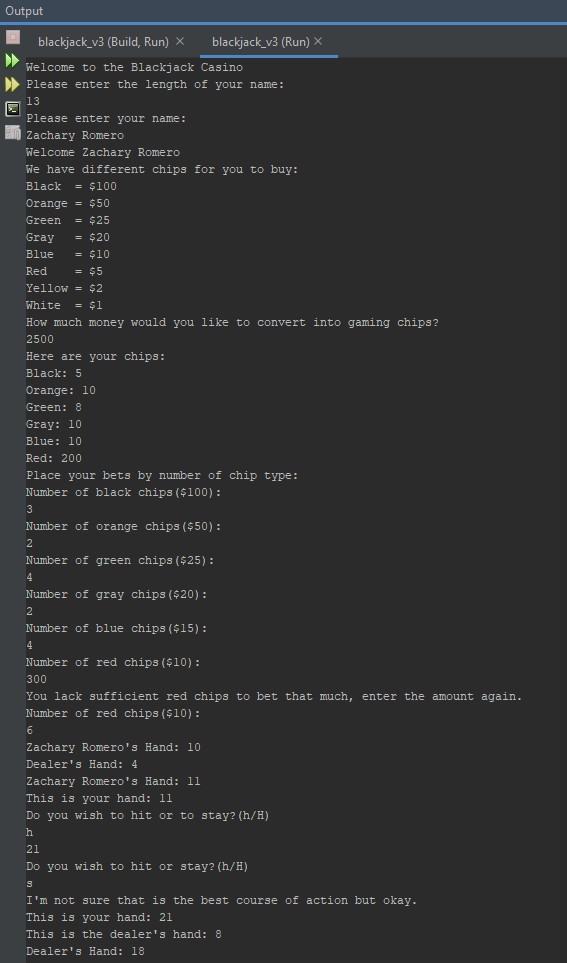
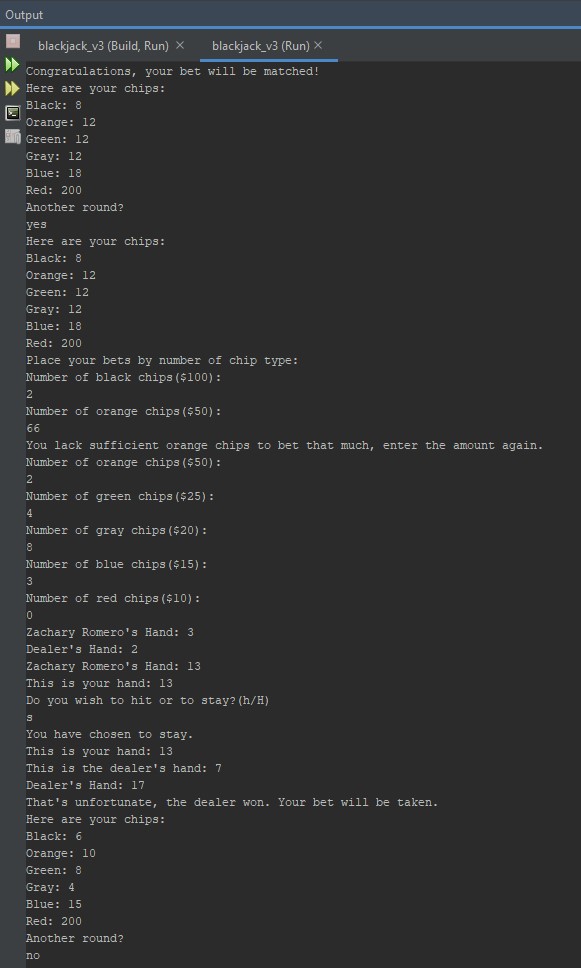
Summary

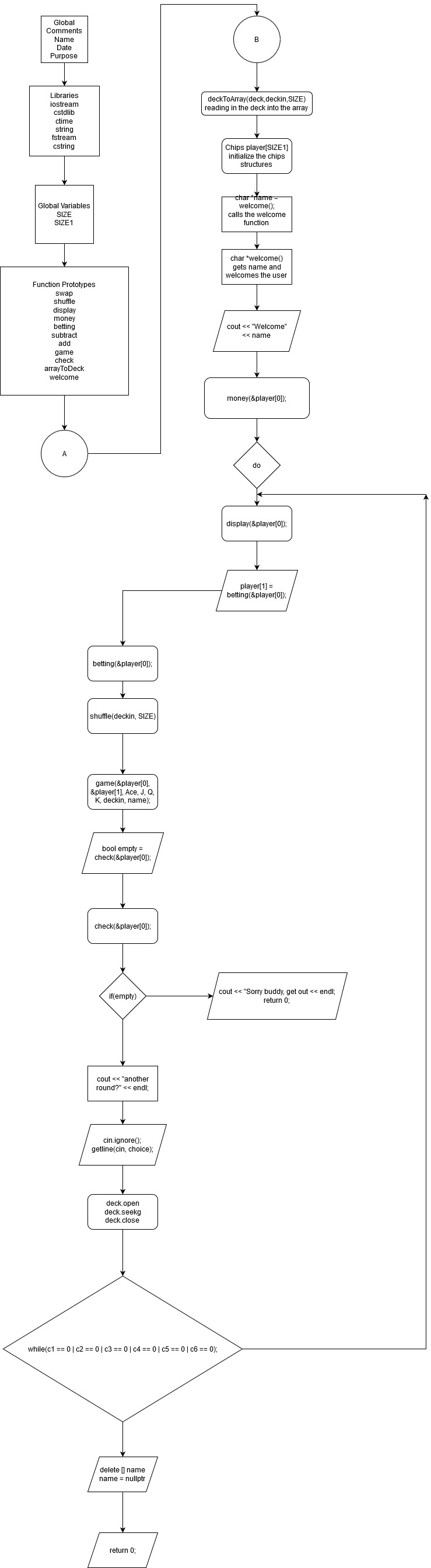
The program is 446 lines long and meets most of the criteria for the project. It is missing I think file formatting, I/O simultaneous, enumerators (which I thought would have been easy to implement) and nested structures. It has 17 variables, 12 functions, 1 structure, and includes the iostream, cstdlib, ctime, string, fstream, and cstring libraries.

I found this project to be challenging at first, it slowly became easier as I found my groove with what I wanted to do and then slowly became much more difficult once I realized that I needed to have more concepts in the project than I was naturally using. I guess it is a decent tell as to where I am at in terms of understanding the C++ language and coding so far. I did not use much if anything outside of what was given by the book from what I can remember after working on the whole thing for about two weeks now on and off.

There is an issue I have noticed but have not been able to replicate, but the deck.dat file seems to occasionally make the program not run. I have made sure to close in all the relevant places, deallocate memory and the like but still for some reason does not run sometimes. In the case this happens while you grade my program, there is a deck.dat file in the main file. If you want, replace the deck.dat file in the blackjack folder with the project deck.dat file.

Description

Sample Input/Output

Flowchart

Pseudocode

//swap function, used for the shuffle function

//shuffle function shuffles the deck

//displays the chips

//distributes chips based on how much money the user decides to put in

//returns a struct for the betting values

//subtracts the betting chips from player chips if they lose

//matches the betting chips from player chips if they win

//the actual blackjack game

//checks if the player is out of money

//my thought process for this was to have the counter be the individual cards in the array

//this is to simulate the player looking at their hand

//this is to simulate the dealer revealing the second card

//if the dealer has a hand less than 17 they must hit until they have 17 or more

//set random number seed

//goes from the end of the array to the beginning

//j becomes a random element in the array from 0 to i

//this calls the swap function

//maybe make different colors different chips with different values

//this section gives the amount of chips per category and stores them in the structure above

//tried to distribute the chips so you'd have more cheap chips rather than expensive ones

//this function is going to take in the number of chips per type for each round of the game

//it will also verify that the player has enough chips to bet

//it will then subtract the amount from the player's struct

//subtracts when bet is lost

//adds when bet is won

//simply displays the chips

//adds up all the chips and sees if they are out of chips

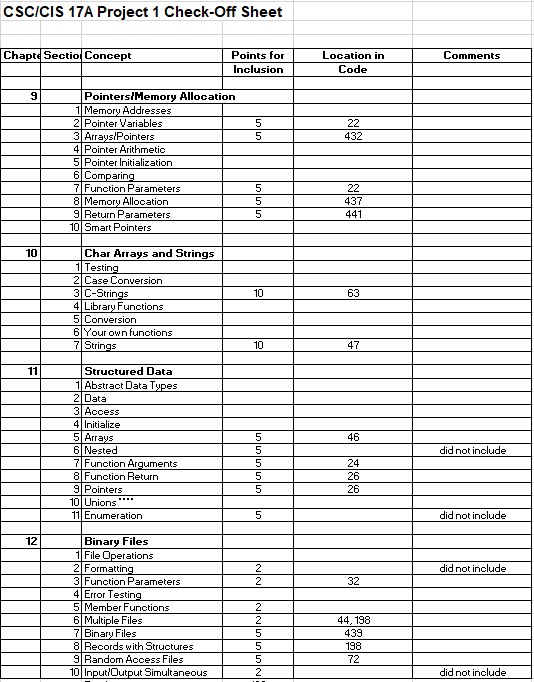
//welcome function, when i ran out of ideas on how to incorporate more of the needed concepts in the project

//not sure if this is legal but i ran with it anyway

//i know, i can't think of how to put this into the game

Variables

|  |  |
| --- | --- |
| Bool empty = false | Line 47 |
| Fstream deck | Line 50 |
| String choice, choice1-6 | Line 52 |
| Int c1-6 | Lines 72-77 |
| Char name | Line 55 |
| Int hand | Line 97, 120 |
| Int dealer | Line 102, 138 |

Concepts

**Program**

/\*

\* Name: Zachary Romero

\* Date: 24 October 2020 7:40PM

\* Purpose: End blackjack program version, trying to incorporate all of the concepts after the initial base game was done

\*/

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <string>

#include <fstream>

#include <cstring>

using namespace std;

struct Chips

{

int black{};

int orange{};

int green{};

int gray{};

int blue{};

int red{};

int yellow{};

int white{};

};

const int SIZE = 52;

const int SIZE1 = 2;

void swap(int \*, int \*); //swap function, used for the shuffle function

void shuffle(int [], int); //shuffle function shuffles the deck

void display(Chips \*); //displays the chips

void money(Chips \*); //distributes chips based on how much money the user decides to put in

Chips betting(Chips \*); //returns a struct for the betting values

void subtract(Chips \*, Chips\*); //subtracts the betting chips from player chips if they lose

void add(Chips \*, Chips \*); //matches the betting chips from player chips if they win

void game(Chips \*, Chips \*, int, int, int, int, int [], char \*); //the actual blackjack game

bool check(Chips \*); //checks if the player is out of money

void deckToArray(fstream &, int [], const int);

char \*welcome();

int main()

{

int Ace = 1;

int J = 10;

int Q = 10;

int K = 10;

bool empty = false;

fstream deck;

int deckin[SIZE];

deckToArray(deck,deckin,SIZE);

Chips player[SIZE1];

string choice, choice1 = "Yes", choice2 = "yes", choice3 = "Y", choice4 = "y", choice5 = "Again", choice6 = "again";

int c1,c2,c3,c4,c5,c6;

char \*name = welcome();

cout << "Welcome " << name << endl;

money(&player[0]);

do{

display(&player[0]);

player[1] = betting(&player[0]);

shuffle(deckin, SIZE);

game(&player[0], &player[1], Ace, J, Q, K , deckin, name);

empty = check(&player[0]);

if(empty)

{

cout << "Sorry buddy, you're out of money. Get out of the casino." << endl;

return 0;

}

cout << "Another round?" << endl;

cin.ignore();

getline(cin, choice);

c1 = choice.compare(choice1);

c2 = choice.compare(choice2);

c3 = choice.compare(choice3);

c4 = choice.compare(choice4);

c5 = choice.compare(choice5);

c6 = choice.compare(choice6);

deck.open("deck.dat", ios::in | ios::binary);

deck.seekg(0L, ios::beg);

deck.close();

}while(c1 == 0 | c2 == 0 | c3 == 0 | c4 == 0 | c5 == 0 | c6 == 0);

delete [] name;

name = nullptr;

return 0;

}

void game(Chips \*player, Chips \*bet, int Ace, int J, int Q, int K, int deck[], char \*name)

{

int hand = 0;

int dealer = 0;

int counter = 4;

int choice;

for(int i = 0; i < 4; i++)

{

if(i % 2 == 0)

{

hand += deck[i];

counter++;

}

else

{

dealer += deck[i];

counter++;

}

if(i == 0 || i == 2)

cout << name << "'s Hand: " << hand << endl;

if(i == 1)

cout << "Dealer's Hand: " << deck[1] << endl;

}

char choice1;

cout << "This is your hand: " << hand << endl;

cout << "Do you wish to hit or to stay?(h/H)" << endl;

cin >> choice1;

if((choice1 == 'h') || (choice1 == 'H'))

{

do

{

hand += deck[counter]; //my thought process for this was to have the counter be the individual cards in the array

counter++;

cout << hand << endl;

if(hand > 21)

{

cout << name << " busted, better luck next time!" << endl;

return;

}

cout << "Do you wish to hit or stay?(h/H)" << endl;

cin >> choice1;

if(hand >= 17)

cout << "I'm not sure that is the best course of action but okay." << endl;

}while(choice1 == 'h' || choice1 == 'H');

}

else

cout << "You have chosen to stay." << endl;

cout << "This is your hand: " << hand << endl; //this is to simulate the player looking at their hand

cout << "This is the dealer's hand: " << dealer << endl; //this is to simulate the dealer revealing the second card

while(dealer < 17) //if the dealer has a hand less than 17 they must hit until they have 17 or more

{

dealer += deck[counter];

counter++;

cout << "Dealer's Hand: " << dealer << endl;

}

if(hand > dealer)

{

cout << "Congratulations, your bet will be matched!" << endl;

add(player,bet);

display(player);

return;

}

else if(hand == dealer)

{

cout << "That is a push." << endl;

display(player);

return;

}

else if(dealer > 21)

{

cout << "The dealer busted, congratulations your bet will be matched!" << endl;

add(player,bet);

display(player);

return;

}

else

{

cout << "That's unfortunate, the dealer won. Your bet will be taken." << endl;

subtract(player,bet);

display(player);

return;

}

}

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void shuffle(int array[], int num)

{

//set random number seed

srand(static\_cast<unsigned int>(time(0)));

for(int i = num - 1; i > 0; i--) //goes from the end of the array to the beginning

{

int j = rand() % (i + 1); //j becomes a random element in the array from 0 to i

swap(&array[i], &array[j]); //this calls the swap function

}

}

void money(Chips \*player)

{

int totChips{}; //maybe make different colors different chips with different values

int bet{};

cout << "We have different chips for you to buy: " << endl;

cout << "Black = $100" << endl;

cout << "Orange = $50" << endl;

cout << "Green = $25" << endl;

cout << "Gray = $20" << endl;

cout << "Blue = $10" << endl;

cout << "Red = $5" << endl;

cout << "Yellow = $2" << endl;

cout << "White = $1" << endl;

cout << "How much money would you like to convert into gaming chips?" << endl;

cin >> totChips;

fstream chips("chips.dat", ios::out | ios::binary);

while(totChips >= 100) //this section gives the amount of chips per category and stores them in the structure above

{

player->black += 1;

totChips -= 100;

if(player->black == 5) //tried to distribute the chips so you'd have more cheap chips rather than expensive ones

break;

}

while(totChips >= 50)

{

player->orange += 1;

totChips -= 50;

if(player->orange == 10)

break;

}

while(totChips >= 25)

{

player->green += 1;

totChips -= 25;

if(player->green == 8)

break;

}

while(totChips >= 20)

{

player->gray += 1;

totChips -= 20;

if(player->gray == 10)

break;

}

while(totChips >= 10)

{

player->blue += 1;

totChips -= 10;

if(player->blue == 10)

break;

}

while(totChips >= 5)

{

player->red += 1;

totChips -= 5;

}

while(totChips >= 2)

{

player->yellow += 1;

totChips -= 2;

}

while(totChips >= 1)

{

player->white += 1;

totChips -= 1;

}

chips.write(reinterpret\_cast<char \*>(&player), sizeof(player));

chips.close();

}

Chips betting(Chips \*player) //this function is going to take in the number of chips per type for each round of the game

{ //it will also verify that the player has enough chips to bet

Chips bet; //it will then subtract the amount from the player's struct

cout << "Place your bets by number of chip type: " << endl;

if(player->black > 0)

{

do{

cout << "Number of black chips($100): " << endl;

cin >> bet.black;

if(bet.black > player->black)

cout << "You lack sufficient black chips to bet that much, enter the amount again." << endl;

if(bet.black < 0)

cout << "Negative value, try again." << endl;

}while(bet.black > player->black || bet.black < 0);

}

if(player->orange >0)

{

do{

cout << "Number of orange chips($50): " << endl;

cin >> bet.orange;

if(bet.orange > player->orange)

cout << "You lack sufficient orange chips to bet that much, enter the amount again." << endl;

if(bet.orange < 0)

cout << "Negative value, try again." << endl;

}while(bet.orange > player->orange || bet.orange < 0);

}

if(player->green > 0)

{

do{

cout << "Number of green chips($25): " << endl;

cin >> bet.green;

if(bet.green > player->green)

cout << "You lack sufficient green chips to bet that much, enter the amount again." << endl;

if(bet.green < 0)

cout << "Negative value, try again." << endl;

}while(bet.green > player->green || bet.green < 0);

}

if(player->gray > 0)

{

do{

cout << "Number of gray chips($20): " << endl;

cin >> bet.gray;

if(bet.gray > player->gray)

cout << "You lack sufficient gray chips to bet that much enter the amount again." << endl;

if(bet.gray < 0)

cout << "Negative value, try again." << endl;

}while(bet.gray > player->gray || bet.gray < 0);

}

if(player->blue > 0)

{

do{

cout << "Number of blue chips($15): " << endl;

cin >> bet.blue;

if(bet.blue > player->blue)

cout << "You lack sufficient blue chips to bet that much enter the amount again." << endl;

if(bet.gray < 0)

cout << "Negative value, try again." << endl;

}while(bet.blue > player->blue || bet.blue < 0);

}

if(player->red > 0)

{

do{

cout << "Number of red chips($10): " << endl;

cin >> bet.red;

if(bet.red > player->red)

cout << "You lack sufficient red chips to bet that much, enter the amount again." << endl;

if(bet.red < 0)

cout << "Negative value, try again." << endl;

}while(bet.red > player->red || bet.red < 0);

}

if(player->yellow > 0)

{

do{

cout << "Number of yellow chips($5): " << endl;

cin >> bet.yellow;

if(bet.yellow > player->yellow)

cout << "You lack sufficient yellow chips to bet that much, enter the amount again." << endl;

if(bet.yellow < 0)

cout << "Negative value, try again." << endl;

}while(bet.yellow > player->yellow || bet.yellow < 0);

}

if(player->white > 0)

{

do{

cout << "Number of white chips($1): " << endl;

cin >> bet.white;

if(bet.white > player->white)

cout << "You lack sufficient white chips to bet that much, enter the amount again." << endl;

if(bet.white < 0)

cout << "Negative value, try again." << endl;

}while(bet.white > player->white || bet.white < 0);

}

//subtracting the betting amounts from the player struct

return bet;

}

void subtract(Chips \*player,Chips \*bet) //subtracts when bet is lost

{

if(bet->black != 0)

player->black -= bet->black;

if(bet->orange != 0)

player->orange -= bet->orange;

if(bet->green != 0)

player->green -= bet->green;

if(bet->gray != 0)

player->gray -= bet->gray;

if(bet->blue != 0)

player->blue -= bet->blue;

if(bet->red != 0)

player->red -= bet->red;

if(bet->yellow != 0)

player->yellow -= bet->yellow;

if(bet->white != 0)

player->white -= bet->white;

}

void add(Chips \*player,Chips \*bet) //adds when bet is won

{

if(bet->black != 0)

player->black += bet->black;

if(bet->orange != 0)

player->orange += bet->orange;

if(bet->green != 0)

player->green += bet->green;

if(bet->gray != 0)

player->gray += bet->gray;

if(bet->blue != 0)

player->blue += bet->blue;

if(bet->red != 0)

player->blue += bet->blue;

if(bet->yellow != 0)

player->yellow += bet->yellow;

if(bet->white != 0)

player->white += bet->white;

}

void display(Chips \*player) //simply displays the chips

{

cout << "Here are your chips: " << endl;

if(player->black != 0)

cout << "Black: " << player->black << endl;

if(player->orange != 0)

cout << "Orange: " << player->orange << endl;

if(player->green != 0)

cout << "Green: " << player->green << endl;

if(player->gray != 0)

cout << "Gray: " << player->gray << endl;

if(player->blue != 0)

cout << "Blue: " << player->blue << endl;

if(player->red != 0)

cout << "Red: " << player->red << endl;

if(player->yellow != 0)

cout << "Yellow: " << player->yellow << endl;

if(player->white != 0)

cout << "White: " << player->white << endl;

}

bool check(Chips \*player) //adds up all the chips and sees if they are out of chips

{

bool empty = false;

int total = player->black + player->orange + player->green + player->gray + player->blue

+ player->red + player->yellow + player->white;

if(total == 0)

empty = true;

return empty;

}

void deckToArray(fstream &in, int deck[], int SIZE) //writes the binary file to the deck

{

in.open("deck.dat", ios::in | ios::binary);

in.read(reinterpret\_cast<char \*>(deck), sizeof(deck) \* SIZE);

in.close();

}

char \*welcome() //welcome function, when i ran out of ideas on how to incorporate more of the needed concepts in the project

{ //not sure if this is legal but i ran with it anyway

int sizeIn;

char \*name = nullptr;

cout << "Welcome to the Blackjack Casino" << endl;

cout << "Please enter the length of your name: " << endl; //i know, i can't think of how to put this into the game

cin >> sizeIn;

sizeIn += 2;

name = new char[sizeIn];

cout << "Please enter your name: " << endl;

cin.ignore();

cin.getline(name, sizeIn);

return name;

}