**Baccarat**

**Card Game**

Jacob Carritte

CIS-5

Spring 2021

43358

TABLE OF CONTENTS

INTRODUCTION 3

HOW THE CARD GAME WORKS 3

MY APPROACH TO THE GAME 3

THE LOGIC OF IT ALL 4

CONSTRUCTS & CONCEPTS UTILIZED 24

PROOF OF WORKING PRODUCT 27

REFERENCES 32

PROGRAM 33

**Introduction**

*“Gambling: The sure way of getting nothing for something.”*

-Wilson Mizner

Baccarat is a card game which originated in mid 19th century France. Also termed Punto Banco, it is a casino game which requires no technical skills. It’s frequently found in high-stakes casinos in which players bet large sums. Though I’ve never played in real life, the game has always interested me and seemed a perfect fit for this project with its reliance on chance and the complexity of the rules.

**How the Card Game Works**

**Objective**

Bet on either banker or player hand for which comes closest to totaling 9, in addition to betting on appearance of pairs or ties.

**Rules of the Game**

Cards are ranked with Face cards and 10s counting as 0 points. All other cards have their correspondingly designated point values. Bets are placed on banker hand winning, player hand winning, or a tie between the two. The payout of the player hand winning is 1:1, banker hand winning is 1:0.95, and an 8:1 payout for ties. Any hand exceeding 10 points receives a 10 point subtraction, and a third card is drawn if the first two total less than 5 points.

**My Approach to the Game**

**Translating Game Play Rules to Programming Language**

The game is primarily reliant on chance, as 6-8 decks are used, therefore, counting cards would not be beneficial. As a result, this made the game easier to code and drawn cards did not need to be accounted for in the same manner. In coding Baccarat, I used the rand function to generate numbers between 1 and 14, translate face card values to a representative string from their original integers, and pair them with a randomized suit generated and translated from integers 1-4. After this, I used mathematics and conditional statements to determine if a third card is to be drawn for both player and banker hands. After the cards and winners have been determined, the initial bets input by the user (or taken from an external file), are calculated based on the results of the match and payouts are determined. I chose to have the player begin with $1000 as a starting sum and records are kept of the player/house winnings and the rounds played. Averages and other game statistics are calculated and output to the console along with records of names, scores (in end $), and rounds played (stored in single/2d arrays) entered into an output file for future referencing. A final function that I created allows the user to search for scores and select that they be sorted by both high scores in descending order and name in alphabetical order.

**Similarities to the Card Game**

The game I coded for is similar to the card game in almost all respects other than minor differences that could potentially affect the statistics if played long-term.

**Differences from the Card Game**

The major differences between my game and Baccarat in real life are that because 6-8 decks are used, likelihood of drawn cards being drawn a second time will slightly decrease. This should be nearly negligible, but my program randomizes cards without any alterations based on those previously drawn, which could potentially impact results. Additionally, players do not start with a default sum of $1000 and high scores are not kept by casinos. Other than these small differences, the games are nearly identical.

**The Logic of it All**

**Flowchart**

My flowchart is multiple pages, so I’ll break it up into smaller pieces with pseudocode accompanying each section.

*Opening comments*

*Bring in system libraries*

*Set function prototypes*

*Enter main*

*Declare all variables, initiating some now and others later*

*Open file to input data*

*Open file to output data*

*Set variables to initial values*

*Output greeting*

*Input name to be identified by*

*Run prnt() function to output personalized greeting*

*Enter into do-while loop*

*Randomize cards using rndCrd() function*

*Enter into for loop iterating through randomization of suit values (1-4)*

*Enter into for loop iterating through translation of suit values into suit names*

*Run prnt() function for output of player and house info*

*Output directions to input bets*

*Input bets*

*Output user-entered bet values*

*Enter into for loop iterating through card integers to generate strings for face cards*

*\*If cards[0] is less than 11 output card and suit or else output string for generated face card name and suit*

*Repeat above process beginning at \* for cards[1]*

Diagram

Description automatically generated

*\*If cards[0] >= 10 set subTot[0] = cards[0] or else subTot[0] = 0*

*Repeat the above step for cards[1] and card[4]*

*If subTot[0] + subTot[1] is less than 5 or greater than 9, check if cards[4] is less than 11 and output proper number or face card or else set subTot[4] equal to 0 if false*

*Run refFunc() function to calculate total for player*

*If player total is greater than 9 subract 10 or else leave as is*

*Output player points*

*Repeat process beginning at last \* for cards[2], cards[3], and cards[5], respectively calculating banker totals and output results (process continues onto next page)*Diagram

Description automatically generated

*Check for pair within player and banker hands and set Booleans accordingly*

Diagram

Description automatically generated

*Check if player total is greater than bank total and set values of payOut array accordingly*

*Check if player hand has pairs and add winnings if true or else subtract player bets*

*Check if bank hand has pairs and add winnings if true or else subtract corresponding bets*

*Output winner and money won*

*If false, check if banker total is greater than player total and set values of payOut array accordingly*

*Check if player hand has pairs and add winnings if true or else subtract player bets*

*Check if bank hand has pairs and add winnings if true or else subtract corresponding bets*

*Output winner and money won*

Diagram, engineering drawing

Description automatically generated

*If false, check if banker total is equal to player total and set values of payOut array accordingly*

*Check if player hand has pairs and add winnings if true or else subtract player bets*

*Check if bank hand has pairs and add winnings if true or else subtract corresponding bets*

*Output tie notification and money won*

*Run hPay function to calculate new player and house money totals and output*

*Ask player if they would like to play again and check user input*

*If appropriate user input (y, Y, or n, N), increment number of rounds played and repeat do-while loop if y or Y*

*If invalid input, reprompt*

*If user enters n or N, close input file and run endGame() function to output game info*

*Check if output file is open and write in Name, end $, and number of rounds played if true or else state that file cannot be opened*

*Close output file*

*Run viewScr() function to check high scores, sort, and perform binary search*

*End main*

*Diagram

Description automatically generated*

*Enter viewScr function*

*Declare variables and set values of some*

*Output prompt to view high scores*

*User input response*

*If invalid input prompt to reenter y/n*

*Accept secondary input from user*

*Or else, if appropriate input and yes, open input file*

*Run for loop iterating to size of array and perform getline() function to take in names, scores, and rounds from file*

*Run selSrt() function for selection sort in descending order of high scores*

*Iterate through, outputting names and scores*

*Output title and description then iterate through outputting names, scores, and rounds*

*Run a binary search for scores through the function scrSrch()*

*Prompt user to enter y/n to determine whether to sort scores alphabetically*

*Accept user input*

*Check if valid*

*If invalid reprompt*

*If valid and yes, run bublSrt() function to perform a bubble sort based on ASCII value of first letter*

*Output title and description*

*Iterate through and output names, scores, and rounds in new order*

*Run binary search based on scores*

*If user input invalid, output that user selected no and run score search function*

*Output goodbye and end function*

*If false, output alert and end function*

*Diagram, engineering drawing

Description automatically generated*

*Enter rndCrd function*

*Declare variables*

*Iterate through based on size of vector and if true randomize card values*

*End function when false*

*Enter rndSuit function*

*Declare variables*

*Set variable as randomly generated number between 1 and 4*

*Return number*

*End function*

*Enter faceGen function*

*Declare variables*

*If card value is 11 or greater, enter switch statement*

*If 11, set variable as “Jack”*

*If 12, set variable as “Queen”*

*If 13, set variable as “King”*

*If 14, set variable as “Ace”*

*If false, return variable and end function*

Diagram

Description automatically generated

*Enter suitGen function*

*Declare variables*

*Enter switch statement*

*For index value 1, set suit to “Hearts”*

*For 2, set suit to “Clubs”*

*For 3, set suit to “Diamonds”*

*For 4, set suit to “Spades”*

*Return suit and end function*

*Enter endgame function*

*Declare variables*

*Output goodbye message*

*Enter for loop, iterating based on number of rounds and output X for each if statement is true*

*If false, calculate average winnings and output the average profit per game*

*End function*

*Enter bublSrt function*

*Declare variables*

*Enter for loop iterating through based on size of array-1*

*If true, enter secondary for loop iterating through based on index value of size – first index value – 1*

*If true compare nms[j] to nms[j+1]*

*If greater, swap values around to render condition false*

*End function*

*Diagram

Description automatically generated*

*Enter selSrt function*

*Declare variables*

*Enter for loop and iterate based upon statement of index < size of array – 1*

*If true, iterate through second for loop with j=i+1 and statement for j<size of array*

*If true compare array[j] to array[i]*

*If a[j] is greater*

*Swap index values of three ways for each array*

*If false, end function*

*Enter binSrch function*

*Declare variables*

*Enter for loop and compare i to size of arrays*

*If true, set Boolean to true and index = I and output money at time of completion and index position*

*Run exit function and end function*

*If false, return found and end function*

*Enter hPay function*

*Declare variables*

*Set variable equal to total of all elements of pay array added together and return variable*

*End function*

*Enter refFunc*

*Declare variables*

*Add up subtotals and set equal to reference variable*

*End function*

*Diagram

Description automatically generated*

*Enter prnt function*

*Declare variables with default string if string argument*

*Output player money and house money if float argument*

*Output player greeting if string argument*

*End function*

*Enter scrSrch function*

*Declare variables*

*Output prompt asking if player wants to run search*

*Accept user input*

*If invalid input, reprompt and reaccept input*

*If valid and yes, output prompt to enter score to search for*

*Accept user input*

*Run binary search function and set equal to Boolean variable*

*If variable is true, output that the score is not found*

*If false, end function*

*If ‘n’ or ‘N’ (y or Y is false), output “You entered “No”” and end function*

*Diagram

Description automatically generated*

**Constructs & Concepts Utilized**

**iostream Library**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| static\_cast | 1 | Statically cast as a different variable | Line 494 |
| cout | 51 | Output Data | Throughout |
| cin | 11 | Input Data | Throughout |
| getline() | 1 | Reads string data | Line 377 |

**iomanip Library**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| setw() | 7 | Set width of output | Throughout |
| setprecision() | 2 | Set number of decimal places/Format floating-point values for output | Lines 574 and 161 |
| fixed | 2 | Sets floatfield format flag for strings to fixed point notation | Lines 574 and 161 |

**stdlib.h Library**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| rand() | 13 | Generates random number | Throughout |
| srand() | 1 | Initializes random number generator | Line 81 |

**time.h/ctime Library**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| time() | 1 | Get current time (for random number generation) | Line 81 |

**string Library**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| string | 27 | Loads string type availability (unnecessary with current version) – declare variable | Throughout |
| getline() | Previously mentioned | Previously mentioned | Previously mentioned |

**fstream Library**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| out.open() | 2 | Opens file for manipulation | Line 120 and 121 |
| out.close() | 2 | Closes file | Line 343 and 332 |

**cmath Library**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| round() | 1 | Returns integer value of nearest whole number | Line 495 |

**vector Library**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| vector<> | 6 | Sequence container type representing arrays that can change in size | Throughout |

**Data Types:**

|  |  |  |
| --- | --- | --- |
| Data Types | Frequency | Location |
| unsigned short | 21 | Throughout |
| bool | 16 | Throughout |
| string | 24 | Throughout |
| const int | 11 | Throughout |
| float | 16 | Throughout |
| char | 4 | Throughout |
| time\_t | 1 | Line 111 |
| vector | 3 | Line 113, 425, 438 |

**Conditional Statements:**

|  |  |  |
| --- | --- | --- |
| Conditional Statement | Frequency | Starting Location |
| if | 1 | Line 440 |
| if/else | 15 | Line 176, 182, 195, 196, 215, 221, 233, 234, 263, 269, 281, 287, 336, 374, 403 |
| if/else if | 1 | Line 259 |
| switch | 2 | Line 441, 465 |
| Ternary operator | 10 | Line 190-192, 211, 228-230, 249, 253, 256 |

**Loops:**

|  |  |  |
| --- | --- | --- |
| Loops | Frequency | Starting Location |
| for | 14 | Line 139, 143, 172, 376, 385, 390, 406, 426, 290, 504, 505, 526, 525, 545 |
| while | 4 | Line 323, 369, 398, 585 |
| do-while | 1 | Line 136 |

**Proof of a Working Product**

In the event that my program encounters errors and does not function once turned in to Dr. Lehr, I have provided screenshots as proof that the program did work.

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**A computer screen shot

Description automatically generated with low confidence**

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

**References**

1. Dr. Lehr’s Lectures & Lab
2. Starting Out with C++: From Control Structures through Objects” – Gaddiss, Tony. 8th Edition (Textbook)
3. [www.geeksforgeeks.org](http://www.geeksforgeeks.org)
4. [www.w3schools.com](http://www.w3schools.com)
5. [www.programiz.com](http://www.programiz.com)

(Websites used for additional information on multidimensional arrays and vector syntax)

*Baccarat Game Rules*. PlayingCardDecks.com. (n.d.). https://playingcarddecks.com/blogs/how-to-play/baccarat-game-rules.

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

/\*

\* File: main.cpp

\* Author: jakec

\* Project 2, Vesion 9: Baccarat

\* Created on June 4, 2021, 3:05 PM

\*/

//System Libraries

#include <iostream> //Input/Output Library

#include <iomanip> //Use for formatting and setting $ to 10ths place for cents

#include <stdlib.h> //srand, rand, exit function

#include <time.h> //time for random number seed purposes

#include <string> //Displaying cards drawn, etc.

#include <fstream> //To read and write statistics files

#include <ctime> //Print out time

#include <cmath> //Used for rounding to nearest $

#include <vector> //STL Dynamic Array

using namespace std; //Library Name-space

//User Libraries

//Global/Universal Constants -- No Global Variables

//Science, Math, Conversions, Higher Dimensioned constants only

//Function Prototypes

unsigned short rndCrd(vector<unsigned short> &); //Function to randomize card #

unsigned short rndSuit(); //Function to randomize suit - # corresponds to suit

string suitGen(unsigned short [], const int); //Generates suits from corresponding numbers

string faceGen(vector<unsigned short>, int); //Generates face cards as strings from corresponding numbers

bool pairs(unsigned short [],int,int,int); //Determines whether pairs are present and returns true or false

void endGame(unsigned short, float, string); //Outputs average winnings and goodbyes

void bublSrt(int [],int [],string [], const int); //Bubble Sort - repeatedly swaps adjacent elements if misordered

void selSrt(int [],int [],string [],const int); //Selection Sort in ascending order by finding min

bool binSrch(int [], string [], const int, int, int); //Binary search which displays index if found

void viewScr(); //View high scores

float hPay(float [],int,int,int); //Used to calculate new player and house totals

void refFunc(unsigned short &,unsigned short [],int,int,int); //Uses pass by reference to calc bank total points

void prnt(float,float); //Function overloading! Prints player and house money

void prnt(string); //Function overloading! User greeting outputting user input or default

void scrSrch(int [],string [],const int);

//Execution Begins Here

int main(int argc, char\*\* argv) {

//Set the Random number seed

srand(time(NULL));

//Declare variables

const int crdSIZE = 6;

const int paySIZE = 3;

float house, //House Money (Cumulative)

player, //Player money (Cumulative)

betBank, //Bet on banker - 1:0.95

betPlay, //Bet on player - 1:1

betTie, //Bet on tie between banker + player; 8:1 payout

pPair, //Bet on pair drawn for player; 11:1 payout

bPair, //Bet on pair drawn for banker; 11:1 payout

payOut[paySIZE], payTot,

avgWin; //average winnings per round

unsigned short bCards, //Banker cards drawn

pCards, //Player cards drawn

rounds, //rounds played (incremented)

//cards[crdSIZE], //number on cards

suitVal[crdSIZE], //array of randomly generated numbers corresponding to suits

pTot, //Player total

bTot, //Banker total

subTot[crdSIZE]; //Used for calculating zero values for face cards

bool pBool, //Player boolean to check if pair

bBool; //Bank boolean for pair

string pName, //Player name

suits[crdSIZE], //array of suits corresponding to drawn cards

view[crdSIZE], //Face Card display stored in array

statsLn; //Stats line from file

char repeat; //y or n entered by user to indicate repeat

fstream out, //For writing stats in file - output

inBets; //File for taking in bets

time\_t curTime; //Current time

int crdSZ = 6; //Size of array

vector<unsigned short> cards(crdSZ); //Array -number on cards

//Initialize variables

/\*GAME PLAYED WITH 6-8 DECKS OF CARDS SO NO NEED TO MANIPULATE

PROBABILITIES BASED UPON PAST DRAWS\*/

rounds = 0;

inBets.open("inputBets.dat", ios::in); //File to take prewritten bets from

out.open("gamestats.dat", ios::app); //Appended file displaying names and scores

house = 0; //Initial house money

player = 1000; //Initial player money

curTime = time(NULL);

cout << "Welcome to Baccarat!" << endl

<< "The time is " << ctime(&curTime)

<< "Please enter name...";

cin >> pName;

prnt(pName);

//Size of commented out because pointless to include

//cout << "Size of float is " << sizeof(float) << endl;

do {

rndCrd(cards); //Function to generate random values for cards

for (int i=0; i<crdSIZE; i++) {

suitVal[i] = rndSuit(); //Random number generation from 1-4

} //Preparation to be translated into suit

for (int i=0; i<crdSIZE; i++) {

suits[i] = suitGen(suitVal, i); //

}

//Process, map inputs to outputs

//PLACE BETS

prnt(player, house); //Prints out player and house money

cout << "Please input your bets, in dollars, for the following: " << endl;

cout << "Banker, Player, Tie, Player Pair, and Banker Pair, respectively..." << endl;

/\*

Commented out if using file as input...

cin >> betBank >> betPlay >> betTie >> pPair >> bPair;

\*/

//Takes bets from file "inputBets.dat"

inBets >> betBank >> betPlay >> betTie >> pPair >> bPair;

cout << "You placed the following bets:" << endl

<< " Banker: $" << fixed << setprecision(2) << betBank << " at 1:0.95 odds" << endl

<< " Player: $" << betPlay << " at 1:1 odds" << endl

<< " Tie: $" << betTie << " at 8:1 odds" << endl

<< "Player Pair: $" << pPair << " at 11:1 odds" << endl

<< "Banker Pair: $" << bPair << " at 11:1 odds" << endl << endl;

//Display your initial conditions as well as outputs.

//DRAW CARDS FOR PLAYER

//DRAW TWO CARDS FOR BANK AND PLAYER: 10's and face cards counted as 0

//FOR TWO DIGIT NUMBERS, DROP OFF FIRST NUMBER

for (int i=0; i<crdSIZE; i++) {

view[i] = faceGen(cards, i); //function to generate string for corresponding face cards

}

//Output values of cards and suits based on Card #

if (cards[0] < 11) {

cout << "Player Cards:\n" << "Card 1: " << cards[0] << " of " << suits[0] << endl;

}

else {

cout << "Player Cards:\n" << "Card 1: " << view[0] << " of " << suits[0] << endl;

}

if (cards[1] < 11) {

cout << "Card 2: " << cards[1] << " of " << suits[1] << endl;

}

else {

cout << "Card 2: " << view[1] << " of " << suits[1] << endl;

}

//Point value calculation using ternary operators in function

cards[0] >= 10 ? subTot[0] = 0: subTot[0] = cards[0];//Calculates subtotal for card 1

cards[1] >= 10 ? subTot[1] = 0: subTot[1] = cards[1];//Calculates subtotal for card 2

cards[4] >= 10 ? subTot[4] = 0: subTot[4] = cards[4];//Calculates subtotal for card 5

//Determine whether to draw 3rd card

if (subTot[0] + subTot[1] <= 5 || subTot[0] + subTot[1] > 9) {

if (cards[4] < 11) {

cout << "Card 3: " << cards[4] << " of " << suits[4] << endl << endl;

}

else {

cout << "Card 3: " << view[4] << " of " << suits[4] << endl << endl;

}

}

else {

subTot[4] = 0;

cout << endl;

}

//Calculate player points

refFunc(pTot, subTot, 0, 1, 4); //Uses pass by reference to perform player total calc

//pTot = subTot[0] + subTot[1] + subTot[4];

(pTot > 9) ? (pTot = pTot - 10) : (pTot = pTot); //Check if pts > 9 and truncate

cout << "Player Points: " << pTot << endl << endl;

if (cards[2] < 11) {

cout << "Banker Cards:\n" << "Card 1: " << cards[2] << " of " << suits[2] << endl;

}

else {

cout << "Banker Cards:\n" << "Card 1: " << view[2] << " of " << suits[2] << endl;

}

if (cards[3] < 11) {

cout << "Card 2: " << cards[3] << " of " << suits[3] << endl;

}

else {

cout << "Card 2: " << view[3] << " of " << suits[3] << endl;

}

//Point value calculation using ternary operators

cards[2] >= 10 ? subTot[2] = 0: subTot[2] = cards[2];//Calculates subtotal for card 3

cards[3] >= 10 ? subTot[3] = 0: subTot[3] = cards[3];//Calculates subtotal for card 4

cards[5] >= 10 ? subTot[5] = 0: subTot[5] = cards[5];//Calculates subtotal for card 6

//Determine whether to draw 3rd card

if (subTot[2] + subTot[3] <= 5 || subTot[2] + subTot[3] > 9) {

if (cards[5] < 11) {

cout << "Card 3: " << cards[5] << " of " << suits[5] << endl << endl;

}

else {

cout << "Card 3: " << view[5] << " of " << suits[5] << endl << endl;

}

}

else {

subTot[4] = 0;

cout << endl;

}

//Calculate banker points

refFunc(bTot, subTot, 2, 3, 5); //Uses pass by reference to perform bank total calc

//bTot = subTot[2] + subTot[3] + subTot[5];

(bTot > 9) ? (bTot = bTot - 10) : (bTot = bTot); //Check if pts > 9 and truncate

cout << "Banker Points: " << bTot << endl;

//Check if pairs:

//Player:

(cards[0] == cards[1] || cards[0] == cards[4] || cards[1] == cards[4]) ? pBool = true : pBool = false;

//Banker:

(cards[2] == cards[3] || cards[2] == cards[5] || cards[3] == cards[5]) ? bBool = true : bBool = false;

//Winning Calculations

if (pTot > bTot) {

payOut[0] = betPlay; //Pay for player win

payOut[1] = betBank \* -0.95; //Loss for player win

payOut[2] = betTie \* 0;

if (pBool == true) {

payOut[0] += pPair \* 11;

}

else {

payOut[0] -= pPair;

}

if (bBool == true) {

payOut[0] += bPair \* 11;

}

else {

payOut[0] -= bPair;

}

cout << "Player wins!" << endl << "Winnings: $" << payOut[0] + payOut[1] + payOut[2] << endl << endl;

}

else if (bTot > pTot) {

payOut[0] = betPlay \* -1;

payOut[1] = betBank \* 0.95;

payOut[2] = betTie \* 0;

if (pBool == true) {

payOut[1] += pPair \* 11;

}

else {

payOut[1] -= pPair;

}

if (bBool == true) {

payOut[1] += bPair \* 11;

}

else {

payOut[1] -= bPair;

}

cout << "Banker wins!" << endl << "Winnings: $" << payOut[0] + payOut[1] + payOut[2] << endl;

}

else if (bTot == pTot) {

payOut[0] = betPlay \* -1;

payOut[1] = betBank \* -0.95;

payOut[2] = betTie \* 8;

if (pBool == true) {

payOut[2] += pPair \* 11;

}

else {

payOut[2] -= pPair;

}

if (bBool == true) {

payOut[2] += bPair \* 11;

}

else {

payOut[2] -= bPair;

}

cout << "It's a tie!" << endl << "Winnings: $" << payOut[0] + payOut[1] + payOut[2] << endl;

}

//End of round cumulative monetary totals

house -= hPay(payOut, 0, 1, 2); //Function using pass by value

player += hPay(payOut, 0, 1, 2);

cout << endl << "End of Round Totals:" << endl << "Player: $" << player << endl

<< "House: $" << house << endl;

cout << "Would you like to play again? (y/n)" << endl;

cin >> repeat;

while (repeat != 'y' && repeat != 'n' && repeat != 'Y' && repeat != 'N') {

cout << "Please enter y/n\n"

<< "Would you like to play again? (y/n)";

cin >> repeat;

}

rounds++;

} while (repeat == 'y' || repeat == 'Y'); //Checks condition after running through code

//Only accepts capital or lowercase y to play again

inBets.close(); //Close input file

endGame(rounds, player, pName); //Function for endgame calculations and goodbyes

if (out.is\_open()) {

out << pName << " " << player << " " << rounds << endl; //Write name and end winnings into file "gamestats.dat"

}

else {

cout << "Unable to open output file...";

}

out.close(); //Close output file "gamestats.dat"

viewScr(); //Display high scores, sort by score, binary search for value

//Exit stage right

return 0;

}

void viewScr() {

char view,//Yes/no - view high scores

asc, //Yes/no to view in ascending order using bubble sort

scr; //Yes/no for binary score search

int sScr, //Score to search for

index; //Value to return for binary search

fstream hScrs; //File to be manipulated

int SIZE = 30; //Array size - alterable depending on number of scores to hold

int SIZE2 = 2; //2d array 2nd dimension size

string names[SIZE]; //Names stored in array for sorting

int scores[SIZE]; //Scores stored in array for sorting

int rounds[SIZE]; //Rounds stored in array for sorting

int scrRnd[SIZE][SIZE2]; //2d array

bool fnd; //True or false if score found

cout << "Would you like to view high scores? (y/n)\n";

//display scores based on user input or repeat prompt

cin >> view;

while (view!= 'y' && view!='n' && view!='Y' && view!='N') {

cout << "Please enter y/n\n"

<< "Would you like to view high scores? (y/n)\n";

cin >> view;

}

if (view=='y'|| view=='Y') {

hScrs.open("gamestats.dat", ios::in); //reopen to read in

for (int i=0; i<SIZE; i++) {

getline(hScrs, names[i]);

hScrs >> names[i];

hScrs >> scores[i];

hScrs >> rounds[i];

}

selSrt(scores, rounds, names, SIZE); //Sorts from high to low - converted to sort trio as linked

for (int i=0; i<SIZE; i++) { //2d array initialization for accessibility and convenience

scrRnd[i][0] = scores[i];

scrRnd[i][1] = rounds[i];

}

cout << "High Scores (Descending):" << endl;

for (int j=0; j<SIZE; j++) { //Displays names, scores, and rounds, respectively

cout << setw(10) << names[j] << " " << setw(6) << scrRnd[j][0] << " " << setw(3) << scrRnd[j][1] << endl; //Output using 2d array

//cout << setw(15) << names[j] << " " << scores[j] << " " << rounds[j] << endl; Original output using single dimensional array

}

scrSrch(scores, names, SIZE);

cout << "Would you like to view scores by username in alphabetical order? (y/n)" << endl;

cin >> asc;

while (asc!= 'y' && asc!='n' && asc!='Y' && asc!='N') {

cout << "Please enter y/n\n"

<< "Would you like to view scores by username in alphabetical order? (y/n)" << endl;

cin >> asc;

}

if (asc=='y' || asc=='Y') {

bublSrt(scores, rounds, names, SIZE); //Bubble sort

cout << "High Scores (Alphabetical):" << endl;

for (int k=0; k<SIZE; k++) {

cout << setw(10) << names[k] << " " << setw(6) << scores[k] << " " << setw(3) << rounds[k] << endl; //Single dimensional array output post-bubble sort

}

scrSrch(scores, names, SIZE);

}

else {

cout << "You selected \"No\"...\n" << endl;

scrSrch(scores, names, SIZE);

cout << "from bottom of chart...\n"

<< "Goodbye" << endl;

}

}

else {

cout << "You selected \"No\"..." << endl

<< "Goodbye" << endl;

exit(0);

}

}

unsigned short rndCrd(vector<unsigned short> &crd) {

for (int i=0; i<crd.size(); i++) {

crd[i] = rand()%(14+1-2)+2; //Randomization function for cards

} //Outputs card value and inputs into array through iteration

}

unsigned short rndSuit() {

unsigned short num;

num = rand() % 4 + 1;

return num;

}

string faceGen(vector<unsigned short> crds, int x) {

string face;

if (crds[x] >= 11) { //Independent if to display face cards

switch (crds[x]) { //Conversion from denoted number

case 11:

face = "Jack";

break;

case 12:

face = "Queen";

break;

case 13:

face = "King";

break;

case 14:

face = "Ace";

break;

}

}

return face;

}

string suitGen(unsigned short stVal[], const int n) {

string suit;

switch (stVal[n]) {

case 1:

suit = "Hearts";

break;

case 2:

suit = "Clubs";

break;

case 3:

suit = "Diamonds";

break;

case 4:

suit = "Spades";

break;

}

return suit;

}

void endGame(unsigned short rnds, float pWinngs, string name) {

float avgWin;

cout << endl << "Thank you for playing Baccarat, " << name << "!" << endl

<< endl << "Total Rounds (Denoted by X):" << endl;

for (int i=0; i<rnds; i++) { //Visual representation of rounds played

cout << " X" << endl;

}

avgWin = (pWinngs-1000)/static\_cast<float>(rnds); //Conversion to float using type casting

cout << "Average profit per game (rounded to nearest $): " << round(avgWin) << endl

<< endl << "Check file \"gamestats.dat\" for statistics" << endl;

}

void bublSrt(int a[], int r[], string nms[], const int n) {

int temp; //For swapping

int temp2;

string temp3;

for (int i=0; i<n-1; i++) {

for (int j=0; j<n-i-1; j++) {

if (nms[j]>nms[j+1]) {

temp = a[j];

temp2 = r[j];

temp3 = nms[j];

a[j] = a[j+1];

r[j] = r[j+1];

nms[j] = nms[j+1];

a[j+1] = temp;

r[j+1] = temp2;

nms[j+1] = temp3;

}

}

}

}

void selSrt(int a[], int b[], string c[], const int n) { //Conversion for dual

int temp; //For swapping int 1 in process of sorting

int temp2; //For swapping int 2

string temp3; //For names swapping string input (corresponding names)

for (int i=0; i<n-1; i++) {

for (int j = i+1; j<n; j++) {

if (a[j] > a[i]) { //Sort from high to low for high scores

//Swapping

temp = a[i];

temp2 = b[i];

temp3 = c[i];

a[i] = a[j];

b[i] = b[j];

c[i] = c[j];

a[j] = temp;

b[j] = temp2;

c[j] = temp3;

}

}

}

}

bool binSrch(int a[], string b[], int n, int value, int index) {

bool found = false;

for (int i=0; i<n; i++) {

if (a[i]==value) {

found = true;

index=i;

cout << value << " at completion by " << b[i] << " in position " << index << endl;

exit(0); //Exit if found = true

}

else {

found = false;

}

}

return found;

}

float hPay(float pay[], int a, int b, int c) { //Function utilizing pass by value to calculate winnings

int hCalc;

hCalc = pay[a] + pay[b] + pay[c];

return hCalc;

}

void refFunc(unsigned short& oT, unsigned short sT[], int a, int b, int c) {

oT = sT[a] + sT[b] + sT[c]; //Overall total calc from subtotal input

} //Function utilizing pass by reference to make changes

void prnt(string usrName = "Player 1") { //Function overloading - output with string input

cout << "Get ready, " << usrName << "...\n\n"; //Set with default argument

}

void prnt(float pMon, float hMon) { //Function overloading - output with unsigned short input

cout << "Player money: $" << fixed << setprecision(2) << pMon << endl

<< " House money: $" << hMon << endl << endl;

}

void scrSrch(int scrA [], string nms[], const int n) {

int index;

char scr; //Variable to hold yes/no

int sScr; //User entry for score search

bool fnd; //True or false if score found

cout << "Would you like to search for a score? (y/n)\n";

cin >> scr; //Enter y or Y to run binary search function

while (scr!= 'y' && scr!='n' && scr!='Y' && scr!='N') {

cout << "Please enter y/n\n"

<< "Would you like to search for a score? (y/n)\n";

cin >> scr;

}

if (scr=='y' || scr=='Y') {

cout << "Enter a score to search for...\n";

cin >> sScr;

fnd = binSrch(scrA, nms, n, sScr, index); //runs function to find presence and index of score entered

if (fnd==false) {

cout << "Score not found...\nGoodbye" << endl;

}

}

else {

cout << "You entered \"No\"...\n"

<< "Goodbye\n";

; }

}