**Project 1**

**<Blackjack Game>**

**CIS-17a – 48969**

**Oscar Sandoval**

**November 5, 2017**

**Introduction**

Blackjack

The game of Blackjack is a popular casino game where a player is faced against a dealer, who deals cards from a deck, and compete to see who can get the closest to 21, without going over it. The goal of the game is to see who can get the closest, and thus, the winner is decided based on who has the higher total score, so long as neither player goes over 21. If either the player or the dealer go over 21, the one who went over it will lose the game. This game is a comparative kind of game, where final scores are compared to one another to see who has the higher value within the range of 21, and whoever has the higher value will win the game.

The game itself implements strategy as well as luck, given that random cards are dealt each time, and there is no way to know which card will be dealt next. In the course of the game, the player will initially be given 2 cards, who may then decide whether to “hit” or “stand” based on the value of his cards. The game works in that the total value of all the cards in one’s hand is added and based on that, the player must decide whether to ask for another card to be added to his hand, also known as “hit,” or decide to keep his current hand, also known as “stand,” and end his turn for the dealer to play. With each successive card a player decides to add to his hand, his risk of losing the game by going over 21 increases. However, if the player’s total value in hand is a low number in the range of 21, then he may also lose against the dealer’s hand. Once the player ends his turn, the dealer will then play as he, too, decides whether to “hit” or “stand” based on his score. Whoever gets the closest to 21, without going over it, at the end of both turns wins the game, and in the occasion that both the player and the dealer get the same score at the end, the dealer wins the game.

I decided to program this game because I have previously played the game of Blackjack before and have enjoyed the strategic side of the game, as well as the random assigning of cards that person is given. Due to that, I set out to program this game, thinking that it would be a moderately easy task. However, once having started it, I realized there were more obstacles than I had previously anticipated, and thus, the difficult I found in programming this game increased.

**Summary**

Project Size: Over 600 lines

Number of variables: 9

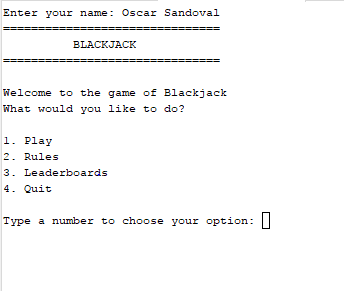
Number of Methods: 12

Number of Structures: 3

In this new revision of my previous game, I included some of the new concepts that we have learned this semester so far, that cleaned up my code a bit as I do not have to be passing around so many variables to functions, which could cause errors if not done carefully. I included structures as well as reading from and writing to binary files, in addition to a variety of other things. Overall, I faced many challenges in trying to improve my game and had to search for a few new concepts that would help me in creating the game I wanted to create. Overall, I believe that I gained a better understanding of all the concepts covered in class and I am now able overcome more obstacles through the higher understanding of programming that I have acquired since I began this game, and if possible, would like to continue to revise this game until it is the best version I can manage to put out. Additionally, I will be able to see my progress in programming through the many versions of this game that I now have.

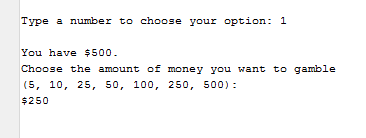
**Screenshots**

**Menu**

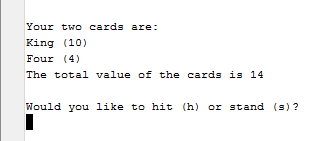


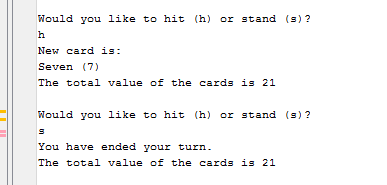
**Option 1: Play**

**Gamble**

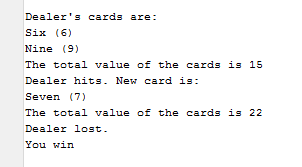


**Hit or Stand**

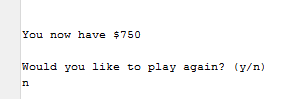




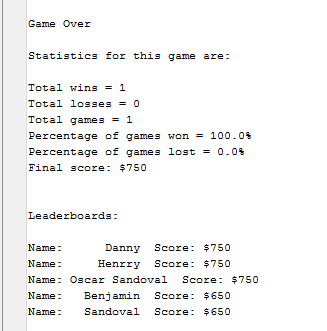
**Dealer’s Turn**



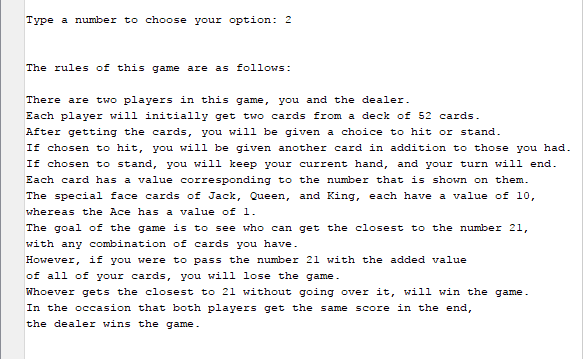
**Play Again?**



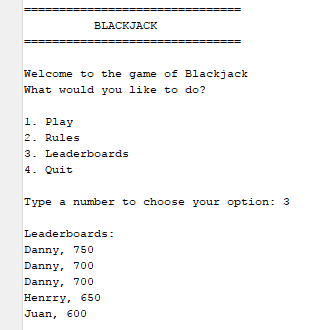
**Game Over**

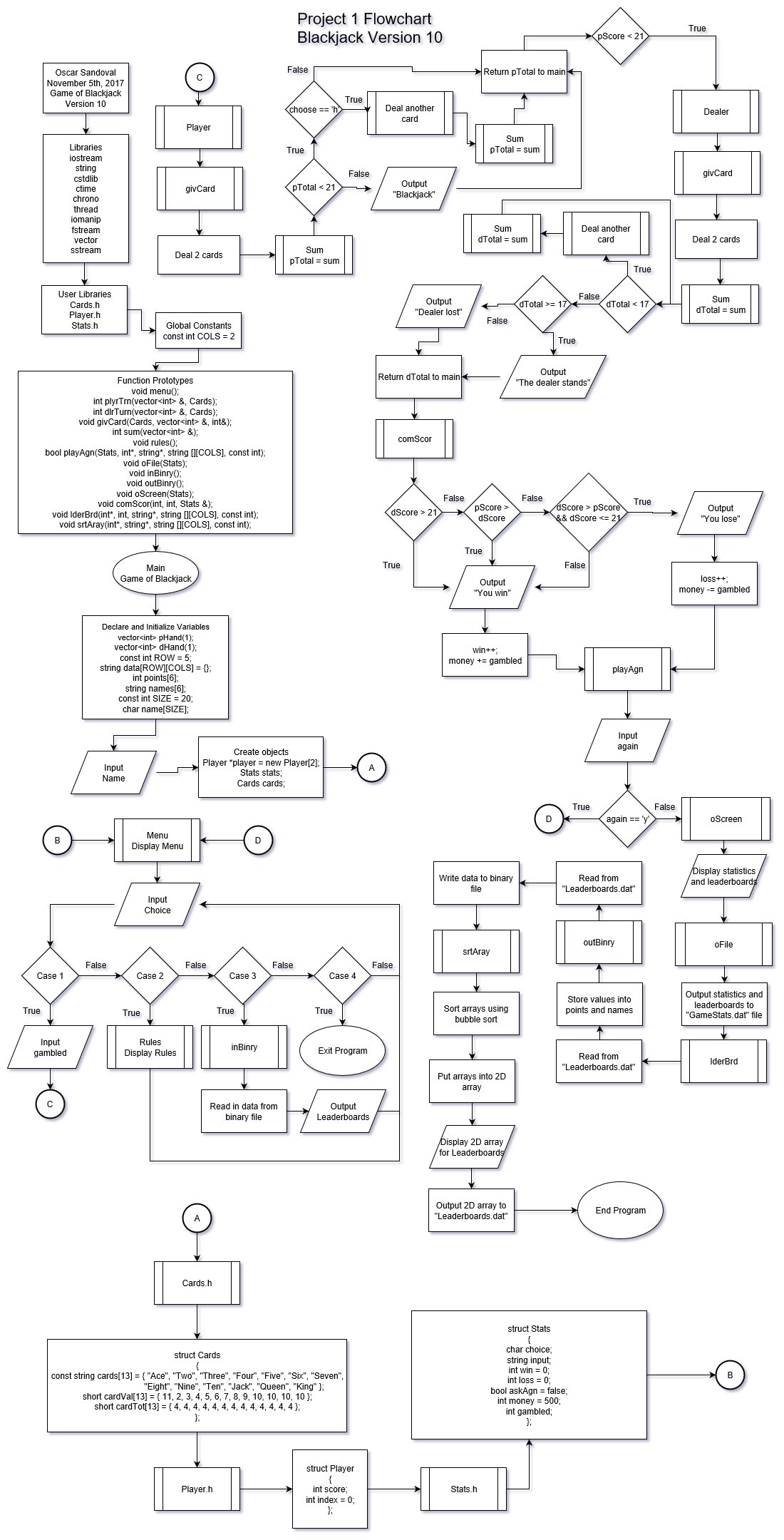


**Option 2: Rules**



**Option 3: Leaderboards**



****

**Cross Reference for Project 1**

**Where in Code**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter** | **Section** | **Topic** | **Line number** |
| 2 | 2 | cout | 378-403 |
|  | 3 | libraries | iostream, iomanip, cmath, cstdlib, fstream, string, ctime |
|  | 4 | variables/literals | 60-72 |
|  | 5 | Identifiers | Yes |
|  | 6 | Integers | 61-69 |
|  | 7 | Characters | 60 |
|  | 8 | Strings | 70 |
|  | 9 | Floats No Doubles | 470, 472 |
|  | 10 | Bools | 65 |
|  | 11 | Sizeof \*\*\*\*\* |  |
|  | 12 | Variables 7 characters or less | Yes |
|  | 13 | Scope \*\*\*\*\* No Global Variables | Yes |
|  | 14 | Arithmetic operators | 452-454 |
|  | 15 | Comments 20%+ | Yes |
|  | 16 | Named Constants | 79 |
|  | 17 | Programming Style \*\*\*\*\* Emulate |  |
|  |  |  |  |
| 3 | 1 | cin | 112 |
|  | 2 | Math Expression | 452-454 |
|  | 3 | Mixing data types \*\*\*\* |  |
|  | 4 | Overflow/Underflow \*\*\*\* |  |
|  | 5 | Type Casting | 470-472 |
|  | 6 | Multiple assignment \*\*\*\*\* |  |
|  | 7 | Formatting output | 447-475 |
|  | 8 | Strings | Yes |
|  | 9 | Math Library |  |
|  | 10 | Hand tracing \*\*\*\*\*\* |  |
|  |  |  |  |
| 4 | 1 | Relational Operators | 293, 301, 310 |
|  | 2 | if | 301-304 |
|  | 4 | If-else | 293-309 |
|  | 5 | Nesting | 356-371 |
|  | 6 | If-else-if | 163-185 |
|  | 7 | Flags \*\*\*\*\* | Yes |
|  | 8 | Logical operators | 142-143 |
|  | 11 | Validating user input | 142-147 |
|  | 13 | Conditional Operator |  |
|  | 14 | Switch | 114-196 |
|  |  |  |  |
| 5 | 1 | Increment/Decrement | 486, 494, 501 |
|  | 2 | While | 523-531 |
|  | 5 | Do-while | 69-197 |
|  | 6 | For loop | 93-100 |
|  | 11 | Files input/output both | 512-558 |
|  | 12 | No breaks in loops \*\*\*\*\*\* |  |
|  |  |  |  |
| 6 | 3 | Function Prototypes | 27-39 |
|  | 5 | Passing by value | 29-39 |
|  | 8 | Returning values from functions | 275, 311, 374 |
|  | 9 | Returning a boolean \*\*\*\*\*\* | 437 |
|  | 10 | No Global Variables Allowed |  |
|  |  | Only Global Constants |  |
|  |  | Meaning Conversions,Physical Con | stants only |
|  | 11 | Static Local |  |
|  | 12 | Default arguments |  |
|  | 13 | Reference Parameters | 29-37 |
|  | 14 | Overloading functions |  |
|  | 15 | Exit function \*\*\*\*\*\*\* | 131 |
|  |  |  |  |
| 7 | 4 | Array Initialization | 44-51 |
|  | 6 | Processing Arrays | 334-346 |
|  | 7 | Parallel Arrays | 574-588 |
|  | 8 | Arrays as function arguments | 29-39 |
|  | 9 | 2-D Arrays | 72, 596-600, 552-553 |
|  | 12 | STL Vector | 54 and 57, 216 |
|  |  |  |  |
| 8 | 1 | Linear and Binary Search | 364-368 |
|  | 3 | Bubble and Selection Sort | 568-580 |
|  | 5 | Search/Sorting Vectors \*\*\*\*\*\* |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| \*\*\*\*\*\* Not | required to | show |  |

Page 1

**Pseudocode**

/\*

\* File: main.cpp

\* Author: Oscar Sandoval

\* Created on June 3rd, 2017, 7:27 PM

\* Purpose: Game of Blackjack Version 7

\*/

//System Libraries

//Input - Output Library

//Needed to use strings

//For rand and srand

//Time for rand

//Format the output

//For I/O Files

//The STL Vector -> Dynamic Array

//For parsing strings

//Name-space under which system libraries exist

//User Libraries

//Global Constants

//Function Prototypes

//Execution begins here

//Array for each card name

//Array for value of each card

//Array for total number of each card value in deck

//Array for player's hand

//Array for dealer's hand

//Declare variables

//Input number to choose option menu

//Total score of player based on his hand's total value

//Total score of dealer based on his hand's total value

//Initialize to 0 each time program loops

//Initialize to 0 each time program loops

//Loops again as long as boolean is false

//Counts the wins of the player

//Counts the losses of the player

//Money given to player to gamble

//Money player wants to gamble

//Choice input by user to gamble

//Number of rows in data array

//2-D Array used to print leaderboards

//For leaderboard

//Declare and initialize array for names

//Ask user to enter his or her name and save to char array name

//Enter player's name into last spot of string array names

//Loop will reset all of the card values in array for player

//and dealer back to zero at the start of each new game

//Resets hand of player back to 0

//Resets hand of dealer back to 0

//Resets all cards of each value back to 4

//Resets boolean back to false each time the program loops

//Resets player's index each time program loops for array to start at the beginning

//Resets dealer's index each time program loops for array to start at the beginning

//Display the main menu

//Choose an option

//Check to see if player meets minimum bet requirements. If not,

//player is not allowed to continue playing and game ends.

//Output the game statistics to screen

// Call function to output statistics to file

//Call function to make leaderboards

//Ask player for amount of money to gamble

//Convert variable back into integer

//Total score of player's hand returned from player function will be assigned to pScore

//If player's score is less than 21, the game continues to dealer's turn

//Total score of dealer's hand returned from dealer function will be assigned to dScore

// Call function to compare player and dealer's scores

//If player's score is greater than 21, the game ends with a loss for the player

//If player's score is equal to 21, player wins

//Display rules of Blackjack

------------------------------------------

// Create menu for game

//Option 1 plays the game

//Option 2 displays the rules of the game

//Option 3 exits the program

------------------------------------------

// Function for player's turn

//Set boolean statement to false to continue loop

//Total sum of player's cards' values

//Choose whether to hit or stand

//Deal the first two cards

//First card dealt will go into first space of array

//Second card dealt will go into second space of array

//Sum of player's hand returned from sum function will be assigned to pTotal

//Loop the question to hit or stand until player stands,

//or wins or loses the game by hitting 21 or over 21

//Reset pTotal back to 0 to give correct sum each time a new card is dealt

//Player inputs choice

//If chosen to hit, another card will be dealt

//Card will go into next space in array

//New sum will be calculated

//Check to see if player has won or lost the game, else continue asking to hit or stand.

//Player has lost. Break out of loop and display a loss

//Loop will continue as long as player chooses to hit

//If chosen to stand, player will end his turn

//Display sum of player's current hand

//Break out of question loop to continue the game

//If option is invalid, continue to ask question until valid option is input

//Loop will continue as long as askAgain is true

//Returns player's total score back to main

----------------------------------------

// Function for dealer's turn

//Total sum of dealer's cards' values

//Deal the first two cards

//First card dealt will go into first space of array

//Second card dealt will go into second space of array

//Sum of dealer's hand returned from sum function will be assigned to dTotal

//If total is less than 17, dealer has to hit.

//Reset dTotal back to 0 to give correct sum

//If total is greater than or equal to 17, dealer has to stand.

//Loop as long as score is less than 17

//Returns dealer's total score back to main

---------------------------------------------------------

// Deals cards to dealer and player and removes them from cards in deck

//Card index used to create random cards and assign a name and value to each

// NOTE: We wait 1000 milliseconds because fast CPUs

// give the same number inside random.

//Set the random number seed.

//Gives out a random card

//Removes the card from the deck.

//Will continue to deal cards as long as there are still cards of that value

---------------------------------------------

// Sum for the value of all cards in your hand

//Returns sum of all cards in hand to dealer and player functions

-----------------------------

// Rules of Blackjack

-----------------------------------------

//Ask player whether to play again or not

//Input character to play again or not

//If chosen not to play, the program will exit

//Output the game statistics to screen

// Call function to output statistics

//Call function to make leaderboards

//If chosen to play, the program will run again

//If option is invalid, question will continue to be asked

------------------------------------------

//Declare and initialize output file

//Output the game statistics to file

--------------------------------------

//Output Statistics to screen

----------------

//Compare player and dealer's scores

//Display the scores of the player and the dealer for comparison

//If dealer's score is greater than 21, player wins

//Player's score must be greater than dealer's score to win

//If dealer's score is greater than player's score, and less than 21, player loses

--------------------------------------------

//Make Leaderboards

//Declare and initialize leaderboards file

//Save names temporarily into strings

//Save scores temporarily into strings

//Read names in file up to the comma and store in name

//Read scores in file after comman and store in score

//Save names into names array

//Convert strings into integers and save to points array

//Increase counter after each loop to completely fill arrays

//Close the file

//Assign player's score to last spot in array to be compared with previous scores

//Call function to sort array for leaderboards

//Declare and initialize output file

//Open the file

//Write the names and scores to file and output the leaderboards to screen

//Close the file

------------------------------------

//Use bubble sort to sort array

//Temporary variable needed to swap scores

//Temporary variable to swap names along with scores

//Set flag to break loop when everything is sorted

//Sort parallel arrays in order from largest to smallest score

//Sort score array in order from largest to smallest

//Sort names of players along with their scores

//Will continue to loop until everything is sorted

//Convert integers to strings

**Program**

/\*

\* File: main.cpp

\* Author: Oscar Sandoval

\* Created on November 5th, 2017, 9:03 PM

\* Purpose: Game of Blackjack Version 10

\*/

//System Libraries

#include <iostream> //Input - Output Library

#include <string> //Needed to use strings

#include <cstdlib> //For rand and srand

#include <ctime> //Time for rand

#include <chrono>

#include <thread>

#include <iomanip> //Format the output

#include <fstream> //For I/O Files

#include <vector> //The STL Vector -> Dynamic Array

#include <sstream> //For parsing strings

using namespace std; //Name-space under which system libraries exist

//User Libraries

#include "Cards.h"

#include "Player.h"

#include "Stats.h"

//Global Constants

const int COLS = 2;

//Function Prototypes

void menu();

int plyrTrn(vector<int> &, Cards);

int dlrTurn(vector<int> &, Cards);

void givCard(Cards, vector<int> &, int&);

int sum(vector<int> &);

void rules();

bool playAgn(Stats, int\*, string\*, string [][COLS], const int);

void oFile(Stats);

void inBinry();

void outBinry();

void oScreen(Stats);

void comScor(int, int, Stats &);

void lderBrd(int\*, int, string\*, string [][COLS], const int);

void srtAray(int\*, string\*, string [][COLS], const int);

//Execution begins here

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

//Vector array for player's hand

vector<int> pHand(1);

//Vector array for dealer's hand

vector<int> dHand(1);

//Declare variables

const int ROW = 5; //Number of rows in data array

string data[ROW][COLS] = {}; //2-D Array used to print leaderboards

//Create arrays for leaderboard names and scores

int points[6]; //Array for players' scores

string names[6]; //Array for players' names

//Declare and initialize array for player's name to be used in leaderboards

const int SIZE = 20;

char name[SIZE];

//Ask user to enter his or her name and save to char array name

cout << "Enter your name: ";

cin.getline(name, SIZE);

//Enter player's name into last spot of string array names

names[5] = name;

//Create objects

Player \*player = new Player[2];

Stats stats;

Cards cards;

do

{

//Loop will reset all of the card values in array for player

//and dealer back to zero at the start of each new game

for (int i = 0; i < 11; i++)

{

pHand[i] = 0; //Resets hand of player back to 0

dHand[i] = 0; //Resets hand of dealer back to 0

}

//Reset each card of each suit back to 4

for (int i = 0; i < 13; i++)

cards.cardTot[i] = 4; //Resets all cards of each value back to 4

stats.askAgn = false; //Resets boolean back to false each time the program loops

player[0].index = 0; //Resets player's index each time program loops for array to start at the beginning

player[1].index = 0; //Resets dealer's index each time program loops for array to start at the beginning

//Display the main menu

menu();

//Choose an option

cout << "Type a number to choose your option: ";

cin >> stats.choice;

switch(stats.choice)

{

case '1':

//Check to see if player meets minimum bet requirements. If not,

//player is not allowed to continue playing and game ends.

if (stats.money < 5)

{

cout << "\n\nInsufficient funds. You cannot place the minimum bet." << endl;

//Output the game statistics to screen

oScreen(stats);

// Call function to output statistics to file

oFile(stats);

//Call function to make leaderboards

lderBrd(points, stats.money, names, data, ROW);

exit(0);

}

bool gamble;

//Ask player for amount of money to gamble

do

{

gamble = false;

cout << "\nYou have $" << stats.money << ".\nChoose the amount of money "

"you want to gamble \n(5, 10, 25, 50, 100, 250, 500):\n$";

cin >> stats.input;

if(stats.input != "5" && stats.input != "10" && stats.input != "25" && stats.input != "50" &&

stats.input != "100" && stats.input != "250" && stats.input != "500")

{

cout << "\nInvalid input. Choose one of the available amounts." << endl;

gamble = true;

}

stats.gambled = atoi(stats.input.c\_str()); //Convert variable back into integer

if(stats.gambled > stats.money)

{

cout << "\nYou do not have enough funds to place this bet.\n"

"Choose an amount that you can afford." << endl;

gamble = true;

}

} while(gamble);

//Total score of player's hand returned from player function will be assigned to pScore

player[0].score = plyrTrn(pHand, cards);

//If player's score is less than 21, the game continues to dealer's turn

if (player[0].score < 21)

{

//Total score of dealer's hand returned from dealer function will be assigned to dScore

player[1].score = dlrTurn(dHand, cards);

// Call function to compare player and dealer's scores

comScor(player[0].score, player[1].score, stats);

}

//If player's score is greater than 21, the game ends with a loss for the player

else if (player[0].score > 21)

{

cout << "\nYou lose.\n\n" << endl;

stats.loss++;

stats.money -= stats.gambled;

}

//If player's score is equal to 21, player wins

else

{

player[1].score = dlrTurn(dHand, cards);

cout << "You win\n\n" << endl;

stats.win++;

stats.money += stats.gambled;

}

playAgn(stats, points, names, data, ROW);

break;

case '2':

rules(); //Display rules of Blackjack

break;

case '3':

inBinry();

break;

case '4':

cout << "Exit the program." << endl;

exit(0);

default:

cout << "Not a valid option." << endl;

}

}while (stats.choice != '4');

//Delete dynamically allocated structure array

delete [] player;

player = 0;

return 0;

}

// Create menu for game

void menu()

{

cout << "===============================" << endl;

cout << " BLACKJACK " << endl;

cout << "===============================" << endl << endl;

cout << "Welcome to the game of Blackjack" << endl;

cout << "What would you like to do?" << endl << endl;

cout << "1. Play" << endl; //Option 1 plays the game

cout << "2. Rules" << endl; //Option 2 displays the rules of the game

cout << "3. Leaderboards" << endl; //Option 3 displays the leaderboards

cout << "4. Quit" << endl << endl; //Option 4 exits the program

}

// Rules of Blackjack

void rules()

{

cout << "\n\nThe rules of this game are as follows:" << endl << endl;

cout << "There are two players in this game, you and the dealer." << endl;

cout << "Each player will initially get two cards from a deck of 52 "

"cards. " << endl;

cout << "After getting the cards, you will be given a choice to hit or "

"stand." << endl;

cout << "If chosen to hit, you will be given another card in addition to "

"those you had." << endl;

cout << "If chosen to stand, you will keep your current hand, and your "

"turn will end." << endl;

cout << "Each card has a value corresponding to the number that is shown "

"on them.\nThe special face cards of Jack, Queen, and King, each have a "

"value of 10, " << endl;

cout << "whereas the Ace has a value of 1." << endl;

cout << "The goal of the game is to see who can get the closest to "

"the number 21,\nwith any combination of cards you have. " << endl;

cout << "However, if you were to pass the number 21 with the added value\n"

"of all of your cards, you will lose the game." << endl;

cout << "Whoever gets the closest to 21 without going over it, will win "

"the game." << endl;

cout << "In the occasion that both players get the same score in the end,\n"

"the dealer wins the game.\n\n" << endl;

}

// Function for player's turn

int plyrTrn(vector<int> &pHand, Cards cards)

{

bool askAgn = false; //Set boolean statement to false to continue loop

int pTotal = 0; //Total sum of player's cards' values

char choose; //Choose whether to hit or stand

int hIndex = 0;

//Deal the first two cards

cout << "\n\nYour two cards are:" << endl;

givCard(cards, pHand, hIndex); //First card dealt will go into first space of array

givCard(cards, pHand, hIndex); //Second card dealt will go into second space of array

pTotal = sum(pHand); //Sum of player's hand returned from sum function will be assigned to pTotal

if (pTotal < 21)

{

//Loop the question to hit or stand until player stands,

//or wins or loses the game by hitting 21 or over 21

do

{

pTotal = 0; //Reset pTotal back to 0 to give correct sum each time a new card is dealt

cout << "\nWould you like to hit (h) or stand (s)?" << endl;

cin >> choose; //Player inputs choice

if (choose == 'h' || choose == 'H') //If chosen to hit, another card will be dealt

{

cout << "New card is:" << endl;

givCard(cards, pHand, hIndex); //Card will go into next space in array

pTotal = sum(pHand); //New sum will be calculated

//Check to see if player has won or lost the game, else continue asking to hit or stand.

if (pTotal > 21)

{

cout << "Your total is greater than 21." << endl;

break; //Player has lost. Break out of loop and display a loss

}

askAgn = true; //Loop will continue as long as player chooses to hit

}

else if (choose == 's' || choose == 'S') //If chosen to stand, player will end his turn

{

cout << "You have ended your turn." << endl;

pTotal = sum(pHand); //Display sum of player's current hand

askAgn = false; //Break out of question loop to continue the game

}

else

{

cout << "Invalid option." << endl;

askAgn = true; //If option is invalid, continue to ask question until valid option is input

}

} while (askAgn); //Loop will continue as long as askAgain is true

}

else if (pTotal == 21)

{

cout << "Blackjack." << endl;

}

return pTotal; //Returns player's total score back to main

}

// Function for dealer's turn

int dlrTurn(vector<int> &dHand, Cards cards)

{

int dTotal = 0; //Total sum of dealer's cards' values

int hIndex = 0;

//Deal the first two cards

cout << "\n\nDealer's cards are:" << endl;

givCard(cards, dHand, hIndex); //First card dealt will go into first space of array

givCard(cards, dHand, hIndex); //Second card dealt will go into second space of array

dTotal = sum(dHand); //Sum of dealer's hand returned from sum function will be assigned to dTotal

do

{

if (dTotal < 17) //If total is less than 17, dealer has to hit.

{

dTotal = 0; //Reset dTotal back to 0 to give correct sum

cout << "Dealer hits. New card is: " << endl;

givCard(cards, dHand, hIndex);

dTotal = sum(dHand);

if (dTotal > 21)

{

cout << "Dealer lost." << endl;

}

}

else if (dTotal >= 17) //If total is greater than or equal to 17, dealer has to stand.

{

cout << "The dealer stands." << endl;

}

} while (dTotal < 17); //Loop as long as score is less than 17

return dTotal; //Returns dealer's total score back to main

}

// Deals cards to dealer and player and removes them from cards in deck

void givCard(Cards cards, vector<int> &hand, int& hIndex)

{

int cIndex; //Card index used to create random cards and assign a name and value to each

bool givOther = false;

// NOTE: We wait 1000 milliseconds because fast CPUs

// give the same number inside random.

this\_thread::sleep\_for(chrono::milliseconds(1000));

//Set the random number seed.

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

do

{

givOther = false;

//Gives out a random card

cIndex = rand() % 13;

hand[hIndex] = cards.cardVal[cIndex];

if(hand[hIndex] == hand[hIndex - 1])

{

givOther = true;

}

//Removes the card from the deck.

cards.cardTot[cIndex] -= 1;

} while (cards.cardTot[cIndex] == -1 || givOther == true);

//Will continue to deal cards as long as there are still cards of that value

cout << cards.cards[cIndex] << " (" << cards.cardVal[cIndex] << ")" << endl;

hIndex++;

}

// Sum for the value of all cards in your hand

int sum(vector<int> &hand)

{

int total = 0;

for (int i = 0; i < 11; i++)

{

total += hand[i];

if(total > 21)

{

for(int i = 0; i < 11; i++)

{

if(hand[i] == 11)

{

hand[i] = 1;

total -= 10;

}

}

}

}

cout << "The total value of the cards is " << total << endl;

return total; //Returns sum of all cards in hand to dealer and player functions

}

bool playAgn(Stats stats, int\* points, string\* names, string data[][COLS], const int ROW)

{

do //Ask player whether to play again or not

{

char again; //Input character to play again or not

cout << "You now have $" << stats.money << endl << endl;

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

cin >> again;

if (again == 'n' || again == 'N') //If chosen not to play, the program will exit

{

//Output the game statistics to screen

oScreen(stats);

// Call function to output statistics

oFile(stats);

//Call function to make leaderboards

lderBrd(points, stats.money, names, data, ROW);

exit(0);

}

else if (again == 'y' || again == 'Y') //If chosen to play, the program will run again

break;

else //If option is invalid, question will continue to be asked

{

cout << "Invalid option." << endl;

stats.askAgn = true;

}

} while (stats.askAgn);

return stats.askAgn;

}

void oFile(Stats stats)

{

//Declare and initialize output file

ofstream out;

char outName[] = "GameStats.dat";

out.open(outName);

//Output the game statistics to file

out << fixed << setprecision(1) << showpoint;

out << "Statistics for last game were:" << endl << endl;

out << "Total wins = " << stats.win << endl;

out << "Total losses = " << stats.loss << endl;

out << "Total games = " << stats.win + stats.loss << endl;

out << "Percentage of games won = " << static\_cast<float>(stats.win) /

(stats.win + stats.loss) \* 100 << "%" << endl;

out << "Percentage of games lost = " << static\_cast<float>(stats.loss) /

(stats.win + stats.loss) \* 100 << "%" << endl;

out << "Final Score: S" << stats.money << endl;

out.close();

}

void oScreen(Stats stats)

{

cout << fixed << setprecision(1) << showpoint;

cout << "\nGame Over" << endl << endl;

cout << "Statistics for this game are:" << endl << endl;

cout << "Total wins = " << stats.win << endl;

cout << "Total losses = " << stats.loss << endl;

cout << "Total games = " << stats.win + stats.loss << endl;

cout << "Percentage of games won = " << static\_cast<float>(stats.win) /

(stats.win + stats.loss) \* 100 << "%" << endl;

cout << "Percentage of games lost = " << static\_cast<float>(stats.loss) /

(stats.win + stats.loss) \* 100 << "%" << endl;

cout << "Final score: $" << stats.money << endl;

}

void inBinry()

{

string input; //temporary string to input information from leaderboards

fstream file; //Declare file to output in binary

//Create character array with size of input string

char array[75] = {};

//Open file for output in binary mode

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

//Read binary file into character array pointer

file.read(array, sizeof(array));

//Close the file

file.close();

cout << endl;

cout << "Leaderboards:" << endl;

//Display contents of character array

for(int i = 0; i < sizeof(array); i++)

cout << array[i];

cout << endl;

}

void outBinry()

{

//Declare and initialize leaderboards file

char scores[] = "Leaderboards.dat";

int count = 0;

ifstream in("Leaderboards.dat");

string input; //temporary string to input information from leaderboards

fstream file; //Declare file to output in binary

//Open file for output in binary mode

file.open("Stats.dat", ios::out | ios::binary);

//Read leaderboards information to string

getline(in, input, '.');

//Create character array with size of input string

char \*ptr = new char[input.length()];

//Copy contents of string into character array

for(int i = 0; i < input.length(); i++)

ptr[i] = input[i];

//Write leaderboards information into Binary file

file.write(ptr, input.length());

//Close file

in.close();

file.close();

//Delete allocated memory

delete [] ptr;

ptr = 0;

}

void comScor(int pScore, int dScore, Stats &stats)

{

//Display the scores of the player and the dealer for comparison

cout << "\nPlayer = " << pScore << endl;

cout << "Dealer = " << dScore << endl;

if (dScore > 21) //If dealer's score is greater than 21, player wins

{

cout << "\nYou win.\n\n" << endl;

stats.win++;

stats.money += stats.gambled;

}

//Player's score must be greater than dealer's score to win

if (pScore > dScore)

{

cout << "\nYou win.\n\n" << endl;

stats.win++;

stats.money += stats.gambled;

}

//If dealer's score is greater than player's score, and less than 21, player loses

else if (dScore > pScore && dScore <= 21)

{

cout << "\nYou lose.\n\n" << endl;

stats.loss++;

stats.money -= stats.gambled;

}

else if (pScore == dScore)

{

cout << "\nYou win.\n\n" << endl;

stats.win++;

stats.money += stats.gambled;

}

}

void lderBrd(int\* points, int money, string\* names, string data[][COLS], const int ROW)

{

//Declare and initialize leaderboards file

char scores[] = "Leaderboards.dat";

int count = 0;

string name; //Save names temporarily into strings

string score; //Save scores temporarily into strings

string line;

ifstream in("Leaderboards.dat");

while(getline(in, line))

{

stringstream iss(line);

getline(iss, name, ','); //Read names in file up to the comma and store in name

getline(iss, score); //Read scores in file after comma and store in score

names[count] = name; //Save names into names array

points[count] = atoi(score.c\_str()); //Convert strings into integers and save to points array

count++; //Increase counter after each loop to completely fill arrays

}

outBinry();

//Close the file

in.close();

//Assign player's score to last spot in array to be compared with previous scores

points[5] = money;

//Call function to sort array for leaderboards

srtAray(points, names, data, ROW);

//Declare and initialize output file

ofstream out;

//Open the file

out.open(scores);

cout << "\n\nLeaderboards:" << endl << endl;

//Write the names and scores to file and output the leaderboards to screen

for(int i = 0 ; i < 5; i++)

{

cout << "Name: " << setw(10) << data[i][0] << " Score: $" << data[i][1] << "\n";

out << data[i][0] << ", " << data[i][1] << "\n";

}

out << "." << endl;

//Close the file

out.close();

}

//Use bubble sort to sort array

void srtAray(int\* points, string\* names, string data[][COLS], const int ROW)

{

bool swap;

int temp; //Temporary variable needed to swap scores

string sTemp; //Temporary variable to swap names along with scores

string score[5] = {};

do

{

swap = false; //Set flag to break loop when everything is sorted

for(int count = 0; count < 5; count++)

{

//Sort parallel arrays in order from largest to smallest score

if(points[count] < points[count + 1])

{

//Sort score array in order from largest to smallest

temp = points[count];

points[count] = points[count + 1];

points[count + 1] = temp;

//Sort names of players along with their scores

sTemp = names[count];

names[count] = names[count + 1];

names[count + 1] = sTemp;

//Will continue to loop until everything is sorted

swap = true;

}

}

} while(swap);

//Convert integers to strings

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

score[i] = to\_string(points[i]);

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

{

data[i][0] = names[i];

data[i][1] = score[i];

}

}