Introduction

Title: Baccarat

Baccarat is a casino card game commonly played in Vegas. The player and the dealer draw two hands, and whichever hand is the closest to the number 9 wins the match. The goal of the game is to have the hand you're betting on, either the player or the dealer, have the best hand out of the two dealt.

A few extra notes for the explanation of the game:

- After the player and the dealer have been dealt two cards, they stand.
- If one player has a score that is less than 5, they are dealt another card.
- The final score is determined by the value of the cards, and if the score surpasses 10, the first digit of the score is removed. (i.e. 10 = 0; 13 = 3)
- A score of 8 or 9 without the score surpassing 10 is called a natural. (i.e. 18 = 8 is not a natural)
- This is a relatively simple casino game which only requires luck. It's simple to pick up and play.

Project Statistics

- Project Size: 11.1 KB
- Versions: 5
- Line Count: 416
- Functions: 11
- Structures: 3
- Variables: 32 (3 global constants, 2 global variables, 27 local variables)

Summary and Discussion

My Baccarat project contains everything that we have discussed up to this point-- structs, binary files, char arrays, strings, and functions that utilize structs. Overall, I am extremely proud of this project and how much I learned while creating it. However, there is one thing that I wanted to include that didn't pan out. That one thing is a dynamically sized array.

At one point in the project, I gave the Hand structure a variable `string *cards`. This string pointer would take a dynamically sized array of the cards that were drawn in the game. However, when I used the `delete[]` function to get rid of the `new` array I had created, it produced nothing but issues for the rest of the program. Artifacts would appear when the `dispLG(lastGame l)`

function was called, getting worse each time a new game was played. There was no way to delete the new array outside of the function. If I deleted it when `play(bool mode)` was called again, the program would still end with one new array not being deleted, still causing a memory leak.

I fixed this by turning `string *cards` into `string cards[10]`, a fixed size.

One of the things I am most proud of is finding out how to read in a struct from a binary file. Because structs use memory padding, trying to read in a binary file that has a struct in it will end in a segmentation fault. This is because using `sizeof(struct)` will read an incorrect amount of memory. I fixed this by reading in the members of the struct one by one.

However proud I may be of this project, there is always more I can improve on. There are variables that are redundant, function names which make no sense, artifacts left over from previous versions that take up space. On top of that, I had to use global variables. Global variables are embarrassing. I will be excised from the coding community forever if they find out! So I would have loved to somehow figure out a way to get rid of the global variables in my code. Of course, I would also have loved to find out how to properly use a dynamically sized array in my Hand structure.

Pseudocode

(since the main function is super simple, I included play and bet functions as well, so you can understand what's actually happening when the game is played)

Initialize program

main

If user already has save data
Continue into the menu
Else
Set up new save data
Return to the menu after save data has been created

Display menu of selections the user can make
If user selects play button
Begin playing baccarat (play)
If user selects bet button
Bet on a match of baccarat (bet)
If user selects last game
Display last game information (dispLG)
If user selects player data
Display player information (dispDat)

If user selects delete save data
Ensure that the user wants to delete their save data
If user confirms
Delete save data
Exit program (1)
If user does not accept
Exit program (1)
If user selects exit game
Save player data to file
Exit program (0)

play

Initialize variables
Clear screen
Do
Draw a card
If that card has been drawn already
Go back to start of do-while loop
Get the value for that card
Add card to discard pile
Increment counters
While player has drawn less than 2 cards and their score is less than 10
Add data to player's Hand structure
Repeat the same process for the dealer
Remove the first digit of each player's score
If player score is higher than dealer score
Player has won
Else If dealer score is higher than player score
Dealer has won
Else
Tie
Fill data from this game into global variable last game
Exit function

bet

Initialize variables
Display menu of possible betting options
If user inputs 1
Bet on player win
If user inputs 2
Bet on dealer win
If user inputs 3
Bet on tie
Input amount of money to wager

Play game of baccarat (run play function)
Check if user has won their bet
If user input 1
    If player won the game
        Reward player with winnings
    Else
        Do not reward player
If user input 2
    If dealer won the game
        Reward player with winnings
    Else
        Do not reward player
If user input 3
    If game resulted in a tie
        Reward player with winnings
    Else
        Do not reward player
Return money

Variables

Type	**Variable Name**	**Description**	**Location**
Integer	ms	A global constant that gives the value of a second in milliseconds.	Global constant
	drawn	A number of the cards drawn	struct Hand
	score	The score of the cards in the hand	
	wins	User's total amount of wins	struct PlayerData void play(bool mode) void dispDat(PlayerData pd) PlayerData newData()
	losses	User's total amount of losses	
	ties	User's total amount of ties	
	betWon	User's total amount of bets won	struct PlayerData

			float bet() void dispDat(PlayerData pd) PlayerData newData()
	betLost	User's total amount of bets lost	
	menuNum	User's menu selection for the switch case statement in main.	main()
	dScore	Dealer's score in a game of Baccarat.	play(bool mode)
	pScore	Player's score in a game of Baccarat.	
	c	Counter of cards drawn. Resets when turns switch.	
	o	Counter for array of drawn cards.	
	betVal	User's menu selection for the switch case statement in bet.	float bet()
	suitNum	A randomly selected number between 0-3.	string drwCard()
	faceNum	A randomly selected number between 0-12	
	value	A corresponding value that varies depending on the card.	int cardVal(string strCard)
String	cards[10]	An array of the cards in a player's hand.	struct Hand
	suit[4]	A constant array of suit names that correspond with a value.	Global constant
	face[13]	A constant array of face card names that correspond with a value.	
	date	A string of the date and time that the last game of Baccarat was played.	struct LastGame
	winner	Name of whoever won last game. (player, banker, tie)	
	dCard[10]	An array of cards in the dealer's hand.	void play(bool mode)

	pCard[10]	An array of cards in the player's hand.	
	outCard[52]	An array of cards that have already been drawn.	
	card	A string that uses the random values given by faceNum and suitNum to get the name of the card.	drwCard()
Char	name[25]	The user's name.	struct PlayerData PlayerData newData() void dispDat(PlayerData pd)
	c	A char that the user inputs to verify whether or not they want to delete their save data.	void delDat(PlayerData pd)
Float	money	The user's money. Used for betting.	struct PlayerData float bet() void dispDat(PlayerData pd)
	gains	The total amount of profit that the user has garnered.	
	wager	The amount of money that the user would like to wager.	float bet()
	ratio	User's win/loss ratio for regular games of Baccarat.	void dispDat(playerData pd)
	betRtio	User's win/loss ratio for matches in which they wagered money.	
Boolean	same	Boolean that initializes to false, switches to true if a repeat card is detected.	bool repeat(int o, string card, string *deck)
Hand	player	The player's hand. Located in LastGame struct	struct LastGame
	dealer	The dealer's hand. Located in LastGame struct	

	pHand	The player's hand (information is filled in during the game)	void play(bool mode)
	dHand	The dealer's hand (information is filled in during the game)	void play(bool mode)
LastGame	lastGame	Global variable that holds information about the previous game.	void play(bool mode) float bet() int main()
PlayerData	pd	Holds all of the user's saved data.	int main() void play(bool mode) float bet() PlayerData newData() void dispDat(PlayerData pd) void delDat(PlayerData pd)
time_t	curTime	Struct inherited from the time.h header file. Used to find time that a game of baccarat was played.	LastGame fillDat(Hand p, Hand d, int pScore, int dScore)

Flowchart

Flowchart is a .pdf file included in the zip file. Either open that up or click **here** to view a version of the .pdf that is hosted online.

Program

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

using namespace std;
```

```
//Constants
const string suit[4] = {" of Clubs", " of Hearts", " of Spades", " of Diamonds"};
const string face[13] = {"Ace", "Two", "Three", "Four", "Five", "Six", "Seven",
"Eight", "Nine", "Ten", "Jack", "Queen", "King"};
unsigned const int ms = 1000000;

//Structs
struct Hand{
 string cards[10];
 int drawn, score;
};

struct LastGame{
 Hand player;
 Hand dealer;
 string date, winner;
};

struct PlayerData{
 float money;
 char name[25];
 int wins, losses, ties, betWon, betLost;
 float gains;
};

//Global variables. They are not destroyed until the program ends.
//I know global variables are bad, but I need to use these in functions which
cannot otherwise access them.
LastGame lastGame;
PlayerData pd;

//Functions
void play(bool mode);
float bet();
string drwCard();
int cardVal(string strCard);
void clear();
void dispLG(LastGame l);
```

```
bool repeat(int o, string card, string *outCard);
LastGame fillDat(Hand p, Hand d, int pScore, int dScore);
PlayerData newData();
void dispDat(PlayerData pd);
void delDat(PlayerData pd);

//Main function
int main() {
 fstream file;

 //cout << "Opening file.";
 file.open("data.dat", ios::in | ios::binary);
 if(file.fail()) {
   pd = newData();
 } else {
   file.read(reinterpret_cast<char *>(&pd.money), sizeof(pd.money));
   file.read(reinterpret_cast<char *>(&pd.name), sizeof(pd.name));
   file.read(reinterpret_cast<char *>(&pd.wins), sizeof(pd.wins));
   file.read(reinterpret_cast<char *>(&pd.losses), sizeof(pd.losses));
   file.read(reinterpret_cast<char *>(&pd.ties), sizeof(pd.ties));
   file.read(reinterpret_cast<char *>(&pd.gains), sizeof(pd.gains));
   cout << "Welcome back to Baccarat, " << pd.name << "!\n";
 }
 file.close();

 int menuNum;

 cout << "Version 5: THE FINAL UPDATE!\n\n";
 do{
   srand(time(NULL));
   cout << "Balance: $" << fixed << setprecision(2) << pd.money << endl;
   cout << "1: [For Fun] Play a game of Baccarat.\n2: [For Glory] Put some money
down!\n3: View post-game statistics.\n4: View your statistics.\n5: Delete save
data.\n6+: Save and Exit\n";
   cin >> menuNum;
   switch(menuNum){
     case 1: play(0);
     break;
     case 2: pd.money = bet();
```

```
      break;
      case 3: dispLG(lastGame);
      break;
      case 4: dispDat(pd);
      break;
      case 5: delDat(pd);
      break;
      default: break;
    }
 }while(menuNum < 5);
 //Return without saving if delDat was called
 if(menuNum == 5){
   return 1;
 }
 //Save by default
 file.open("data.dat", ios::out | ios::binary);
 file.write(reinterpret_cast<char *>(&pd.money), sizeof(pd.money));
 file.write(reinterpret_cast<char *>(&pd.name), sizeof(pd.name));
 file.write(reinterpret_cast<char *>(&pd.wins), sizeof(pd.wins));
 file.write(reinterpret_cast<char *>(&pd.losses), sizeof(pd.losses));
 file.write(reinterpret_cast<char *>(&pd.ties), sizeof(pd.ties));
 file.write(reinterpret_cast<char *>(&pd.gains), sizeof(pd.gains));
 file.close();

 return 0;
}

//Takes "mode" param; 0 = for fun, 1 = betting.
void play(bool mode){
 string dCard[10], pCard[10], outCard[52];
 int dScore = 0, pScore = 0;
 int c = 0, o = 0;    //c = card drawn counter, o = out cards counter
 Hand dHand, pHand;
 //Clear the screen.
 clear();
 //Player goes first.
 do{
   pCard[c] = drwCard();
   //If we find a card that has already been drawn, continue.
```

```
  if(repeat(c, pCard[c], outCard)){
    //cout << "[R] PLAYER: " << pCard[c] << endl;
    continue;
  };
  pScore += cardVal(pCard[c]);
  outCard[o] = pCard[c];
  cout << "PLAYER: " << pCard[c] << " (+" << cardVal(pCard[c]) << ")\n";
  c++, o++;
  usleep(0.2 * ms);
}while (c < 2 || pScore < 5);
//Transfer the score and cards into their struct.
pHand.score = pScore;
pHand.drawn = c;
for(int i = 0; i < c; i++){
  pHand.cards[i] = pCard[i];
}
usleep(ms);
//Dealer goes next.
c = 0;
do{
  dCard[c] = drwCard();
  if(repeat(c, dCard[c], outCard)){
    //cout << "[R] DEALER: " << dCard[c] << endl;
    continue;
  }
  dScore += cardVal(dCard[c]);
  outCard[o] = dCard[c];
  cout << "DEALER: " << dCard[c] << " (+" << cardVal(dCard[c]) << ")\n";
  c++, o++;
  usleep(0.2 * ms);
}while (c < 2 || dScore < 5);
dHand.score = dScore;
dHand.drawn = c;
for(int i = 0; i < c; i++){
  dHand.cards[i] = dCard[i];
}
usleep(ms);
//Calculate final score; numbers 10 or higher get the first digit lopped off.
pScore %= 10;
```

```
 dScore %= 10;
 cout << "\nPlayer's Score: " << pScore << endl;
 cout << "Dealer's Score: " << dScore << "\n\n";
 //If mode == 0, add wins/losses to playerdata.
 if(mode == 0){
   if(pScore > dScore){
     pd.wins++;
   } else if (pScore < dScore){
     pd.losses++;
   } else {
     pd.ties++;
   }
 }

 lastGame = fillDat(pHand, dHand, pScore, dScore);
 cout << "_____\n\n";
}

float bet(){
 float wager;
 int betVal = 0;
 float money = pd.money;

 if(money == 0){
   cout << "Hey, you have no money! Giving a starting value of $1000.\n";
   money = 1000.00;
 }
  do{
   clear();
   cout << "Balance: $" << fixed << setprecision(2) << pd.money << "\n\n";
   cout << left << setw(16) << "1. Player wins" << setw(4) << "[2:1]\n";
   cout << left << setw(16) << "2. Dealer wins" << setw(4) << "[2:1]\n";
   cout << left << setw(16) << "3. Tie" << setw(4) << "[8:1]\n";
   cin >> betVal;
   switch(betVal){
     case 1: cout << "You will win 2x your bet if you win the game.\nHow much
will you put down?: ";
     break;
```

```
      case 2: cout << "You will win 2x your bet if the dealer wins the game.\nHow
much will you put down?: ";
      break;
      case 3: cout << "You will win 8x your bet if the game ends in a tie.\nHow
much will you put down?: ";
      break;
      default: cout << "Invalid number!\n";
      break;
    }
  }while(betVal > 3 || betVal < 1);

  //Make sure that the user can't bet more than they have.
  do{
    cin >> wager;
  }while(wager > pd.money);

  play(1);

  switch(betVal){
    case 1: if(lastGame.winner == "Player"){
      money += wager*2;
      money -= wager;
      pd.gains += (wager*2 - wager);
      pd.betWon++;
      cout << "\nYou've won your bet! Your wager of $" << wager << " has been
DOUBLED! ($" << wager*2 << ")\n";
    } else {
      money -= wager;
      pd.betLost++;
      cout << "\nYou've lost your bet, and with it, your $" << wager << "...\n";
    }
    break;
    case 2: if(lastGame.winner == "Dealer"){
      money += wager*2;
      money -= wager;
      pd.gains += (wager*2 - wager);
      pd.betWon++;
      cout << "\nYou've won your bet! Your wager of $" << wager << " has been
DOUBLED! ($" << wager*2 << ")\n";
```

```
    } else {
      money -= wager;
      pd.betLost++;
      cout << "\nYou've lost your bet, and with it, your $" << wager << "...\n";
    }
    break;
    case 3: if(lastGame.winner == "Tie"){
      money += wager*8;
      money -= wager;
      pd.gains += (wager*8 - wager);
      pd.betWon++;
      cout << "\nYou've won your bet! Your wager of $" << wager << " has been
OCTUPLED! ($" << wager*8 << ")\n";
    } else {
      money -= wager;
      pd.betLost++;
      cout << "\nYou've lost your bet, and with it, your $" << wager << "...\n";
    }
    break;
    default: break;
  }
   return money;
}

string drwCard(){
 string card;
 int suitNum, faceNum;
  faceNum = rand() % 13;
 suitNum = rand() % 4;

 card = face[faceNum] + suit[suitNum];

 return card;
}

int cardVal(string strCard){
 int value = -1;
  for(int i = 0; i < 13; i++){
   for(int j = 0; j < 4; j++){
```

```
      if(strCard == face[i] + suit[j]){
        value = i+1;
        //If the card is a jack, queen, or king, set the value to 0.
        if(value >= 10 && value <= 13){
          value = 0;
        }
      }
    }
  }
  //cout << value << " has been added to their score.\n";
  return value;
}

bool repeat(int o, string card, string *deck){
  bool same = false;
  for(int i = 0; i <= o; i++){
    if(deck[i] == card){
      same = true;
      break;
    }
  }
  return same;
}

LastGame fillDat(Hand p, Hand d, int pScore, int dScore){
  LastGame l;
  time_t curTime;
  time(&curTime);

  l.player = p;
  l.dealer = d;

  if(pScore > dScore){
    cout << "PLAYER WINS!\n\n";
    l.winner = "Player";
  } else if (dScore > pScore){
    cout << "DEALER WINS!\n\n";
    l.winner = "Dealer";
  } else {
```

```
    cout << "TIE!!!\n\n";
    l.winner = "Tie";
  }

  l.date = ctime(&curTime);

  return l;
}

void dispLG(LastGame l){
  clear();
  if(l.date == ""){
    cout << "No previous game found! Go play a match, dummy.\n\n";
    return;
  }

  cout << "=========================\n";
  cout << "Previous Game on " << l.date << endl;
  cout << "Player's Hand:\n";
  cout << left;
  for(int i = 0; i < l.player.drawn; i++){
    cout << setw(5) << l.player.cards[i] << endl;
  }
  cout << "Score: " << l.player.score % 10;
  if(l.player.score == 8 || l.player.score == 9){
    cout << " (Natural)";
  }

  cout << "\n\nDealer's Hand:\n";
  for(int i = 0; i < l.dealer.drawn; i++){
    cout << setw(5) << l.dealer.cards[i] << endl;
  }
  cout << "Score: " << l.dealer.score % 10;
  if(l.dealer.score == 8 || l.dealer.score == 9){
    cout << " (Natural)";
  }
  cout << "\n\nWINNER: " << l.winner << "\n";
  cout << "=========================\n\n";
}
```

```
void clear(){
 cout << "\033[2J\033[1;1H";
}

PlayerData newData(){
 PlayerData pd;

 cout << "Welcome to Baccarat! It must be your first time playing.\n";
 cout << "What would your name be? (One word, please!): ";
 cin >> pd.name;
 cout << "Well, nice to meet you, " << pd.name << "!\n";
 cout << "Since you're new here, how about I give you $1000?\n";
 cout << "On the house, of course. Gamble responsibly!\n(press enter to
continue)\n";
 cin.ignore(); //Need two here because of the cin
 cin.ignore();
 pd.money = 1000.00;
 pd.losses = 0;
 pd.wins = 0;
 pd.gains = 0.0f;
 pd.ties = 0;
 pd.betWon = 0;
 pd.betLost = 0;
 clear();

 return pd;
}

void dispDat(PlayerData pd){
 float ratio;
 float betRtio;
 ratio = float(pd.wins) / float(pd.losses);
 betRtio = float(pd.betWon) / float(pd.betLost);
 clear();
 cout << "=========================\n";
 cout << pd.name << "'s Baccarat Profile\n\n";
 cout << "WINS: " << pd.wins << endl;
 cout << "LOSSES: " << pd.losses << endl;
```

```
 cout << "TIES: " << pd.ties << endl;
 cout << "W/L RATIO: " << ratio << "\n\n";
 cout << "BETS WON: " << pd.betWon << endl;
 cout << "BETS LOST: " << pd.betLost << endl;
 cout << "W/L RATIO: " << betRtio << "\n\n";
 cout << "TOTAL PROFITS: $" << pd.gains << endl;
 cout << "=========================\n\n";
}

void delDat(PlayerData pd){
 char c;
 cout << "Hey, " << pd.name << ", we'll be sad to see you go.\n";
 cout << "Are you sure you want to delete your save data? (Y/N): ";
 cin >> c;
 switch(c){
   case 'y':
   case 'Y': cout << "You got it. Make sure to come back sometime,
understand?\n";
   if(remove("data.dat") != 0){
     cout << "Error deleting file... hehe, maybe it's a blessing in disguise,
eh?\n";
   } else {
     cout << "Save data's all gone! See you around!\n";
   }
   break;
   case 'n':
   case 'N': cout << "Gotcha, your save data is safe. Still gotta end the
program, though. You understand.\n";
   break;
 }
}
```