Project 1

<Casino Bham's BlackJack>

CSC 17A :48983

Amul Bham

Date: 10/25/2015

Title : Casino Bham's BlackJack

BlackJack is a simple card game in which the player competes against the dealer to get the higher hand but not over 21. Whoever goes over 21 loses. The player receives their first two cards and the dealer receives one. Based on the player's card total, they must decide to "hit" or "stay", trying not to go over 21 but still attempting to have a higher hand than the dealer. The dealer is set up in a way where they never "hit" if they are at 17 or higher, therefore it is the player's decision to hit or stay depending on if they think the dealer will bust or have a low card total.

For example: If the player has a hand of 14, they risk the chance of going over 21 on their next card draw. So they might decide to stay, despite having a low card total, in hopes that the dealer will ultimately bust. If they dealer has a 9 for their first card, they will keep drawing cards till they are at 17 or higher.

Basically the dealer is always playing risky, and it is up to the player to outsmart the dealer based on odds and knowing when to be aggressive or passive. Also note that if the player goes over 21, they lose no matter what because they draw their cards first. If the hand ends in a tie, no one wins. If a user has a blackjack (21), the dealer still has a chance of drawing a tie if they also have a 21.

This game provides a very similar experience to playing BlackJack in a casino and is intended to be a stimulating game based on odds. In addition, I attempted to recreate the personal experience of being in a casino by having the "dealer" learn the player's name. I also included code to allow the player to bet money, given its fake, to make the user feel as though there are consequences to losing and to provide a more authentic experience.

Summary:

Project Size : About 500 lines
Number of Variables : 20
Number of Methods : 20- 25
Repository:
*https://github.com/amulbham/CSC-17A/tree/master/proj/BlackJack_Project_*

I utilized nearly all, if not all, of the concepts we have learned throughout the course of the semester. Although blackjack is a simple game in theory, it requires a lot of logic and program decisions to account for all the possible scenarios in which the game can play out (ex: if user chooses to hit, if they score blackjack, if dealer hits, etc), The project is continuation and improvement on a BlackJack game I have been working

on for several months now.The newer concepts from chapters 9 - 12 include the use of structures, pointers, binary file reading and writing, and utilization of character strings throughout the program. With these new concepts and methods learned in class, I tried to streamline and clean up the code to make it more efficient. The most challenging parts of the program were figuring out how to track multiple “accounts” of users so that a player could play, and then have his friend play but on their own account. This would allow the players to have separate balances in game and only are affected when the player who is associated with that account is playing. The most effective way to implement this design was to use a structure to keep track of the player information, and then store that structure in the file in binary. First I did hours of research and reread the chapter in the book to make sure I properly understood the idea and how it worked. Nothing is worse than attempting to implement a method without fully understanding exactly how it works. Problems arose when the player is prompted to pick which account they would like to play on. I needed to make sure that based on the option that was picked, the program only wrote over that block of memory and preserve the other data on the file. I used random memory allocation to implement this, using the seek function of the fstream to place the program at a given point in the binary file based on the user choice. Then, at the end, I placed the file at the beginning of the point in memory and wrote the new balance to the file. This preserved the other user information while writing the new information of the current player to the file. This allowed for users to play separately and not have their game effect the other players user information.

Another challenging part of the program was determining what abstractions I could make to group together information into structures. First I made a structure for the current hand being played, this kept track of the player and dealer card totals and well as their current hands. This was a structure used specifically to keep track of the goings of the current hand. The player structure was created specifically to store the player information into the file including such information as the current balance and their name. This allowed me to easily store the player information as a structure in a binary file, making it easier to read and write back to it. The use of structures also made it easier to pass the entire block of player information as a structure of reference instead of passing the individual variables. This allowed me to edit information about the contents of the player instead of passing individual pieces.

In addition, character strings were utilized as input validation throughout the program as well as storing the player information. Instead of having to validate whether the user input (yes/no questions) was capitalized or not, I simply used the tolower() function to always guarantee the user input was of lowercase characters. I also used the atoi() function to convert the user buy in amount, this allowed the program to accept the input as a character string, and then convert the string into an integer. This

accounted for any random characters such as , and $ the user may have included in the input. Finally, I used the getline() function for the character arrays and set a bound to ensure the user did not enter a value that is greater than the array bounds.

Overall, the implementation of all these new concepts took about a week to fully implement, with tons of hours of debugging. All in all, I believe I fully took advantage of all these new concepts and after a week of research and coding, truly understand them. Binary file reading and writing was one concept I can truly see myself using for future projects and am glad I implemented it in this project as it forced me to understand it versatility.

Sample Input/Output

Example input: Type Amul to continue on this account and play here
OR start a new account with your name
First the Program Determines whether the user is new or continuing

```
Would you like to continue(c) or start a new game(n)? c/n
c
The current players on record are ...
Player 1
Name: Amul
Balance: $62633

Player 2
Name: Sanil
Balance: $11955

Player 3
Name: Adrian
Balance: $13915

Player 4
Name: John
Balance: $6200

Enter the player number of the account you would
like to play on
```

If continuing, the player.txt file is opened and read to the player so they can choose which account they would like to play on.

If the user is new, the program asks for a c-string name, and explains the rules. Stores name into player.txt

```
Would you like to continue(c) or start a new game(n)? c/n
n
                    BlackJack □(°▽°)☞
****************************************************************
Welcome to the Casino Bham!
My name is Amul (□_□) and i will be your dealer today!
Your name and balance will be saved when you are finished....
What is your name anyway?
Jason
Ah! nice to meet you Jason, you have a lovely name
Anyways, Im going to assume you know how to play BlackJack as you would be foolish
to play against me without any experience...
Would you like the rules explained? y/n
```

```
How much would you like to buy in Jason ?
2300
How much would you like to bet?
1200
Okay sounds good, your remaining balance is $1100
```

The program then determines how much the user would like to buy in and how much they would like to bet for the current hand.

The dealer then deals the cards, the first two for the player and one for the dealer. Based the the card values the player is prompted to hit (h) or stay(s).

```
Your first two cards are....
Seven of Cloves
King of Cloves
Jason your hand equates to 17

**********************************************************************
My first card is a....
Seven of Diamonds
My hand so far equates to 7
**********************************************************************
Would you like to hit or stay? h/s
```

```
Would you like to hit or stay? h/s
s
**********************************************************************
And now, the rest of my hand...
Ace of Diamonds
My Hand equates to 18
**********************************************************************
Thank you for playing Jason, but you lose (□_□)
**********************************************************************
Your remaining balance is $1100
**********************************************************************
Do you want to play another hand, or walk away now?
y/n
```

If the user decides to stay, the rest of the dealer hand is outputted, followed by the outcome. The player could lose, draw, or win. The remaining balance is outputted and then a prompt if the user would like to play another hand.

If the user hits, a new card is outputted and is prompted again to hit or stay, if the user hits a 21, he is congratulated.

```
Would you like to hit or stay? h/s
h
Two of Spades
6 is your new card total
**********************************************************
Would you like to hit or stay? h/s
h
Seven of Hearts
13 is your new card total
**********************************************************
Would you like to hit or stay? h/s
h
Eight of Spades
21 is your new card total
**********************************************************
**********************************************************************
Congratulations you got blackjack! □(°▽°)☞
**********************************************************************
```

```
Do you want to play another hand, or walk away now?
y/n
y
How much would you like to bet?
500
Okay sounds good, your remaining balance is $600
************************************************************
```

If the user wants to play again, the game loop repeats, asking the user to enter a new bet

If the user wants to stop playing, he is told of his winnings/losses for the day and the program exits right.

```
Do you want to play another hand, or walk away now?
y/n
n
Looks like you lost $1700 today, better luck next time Jason
```

Pseudocode:

Initialize

Determine if the user is a returning player or a new player
if returning, read name and balance from file
Determine if the user wants to buy in more
else if the player is new
Display the intro and obtain the player name
Save player name to file
Ask how much they would like to buy in

The user is asked to place a bet
else game will remain in limbo until user is ready

Values of a Deck of cards (2-53) are created and then shuffled
if card number goes over 40, deck is reshuffled
else, same deck of cards is used for hand until card number goes over 40

The user then receives their first two cards

The dealer receives their first card

Program calculates the value of each player's hand after each card draw

if the user decides to hit, they will continue to receive cards till they are over 21, they decide to stay, or hit a BlackJack.

else, the dealer receives their next cards, and continues until they are at 17 or over.

If player or dealer bust, opposing player wins
else, program continues.

Then the program determines who won

if user has a higher card total, they win
The user receives twice the amount that they bet

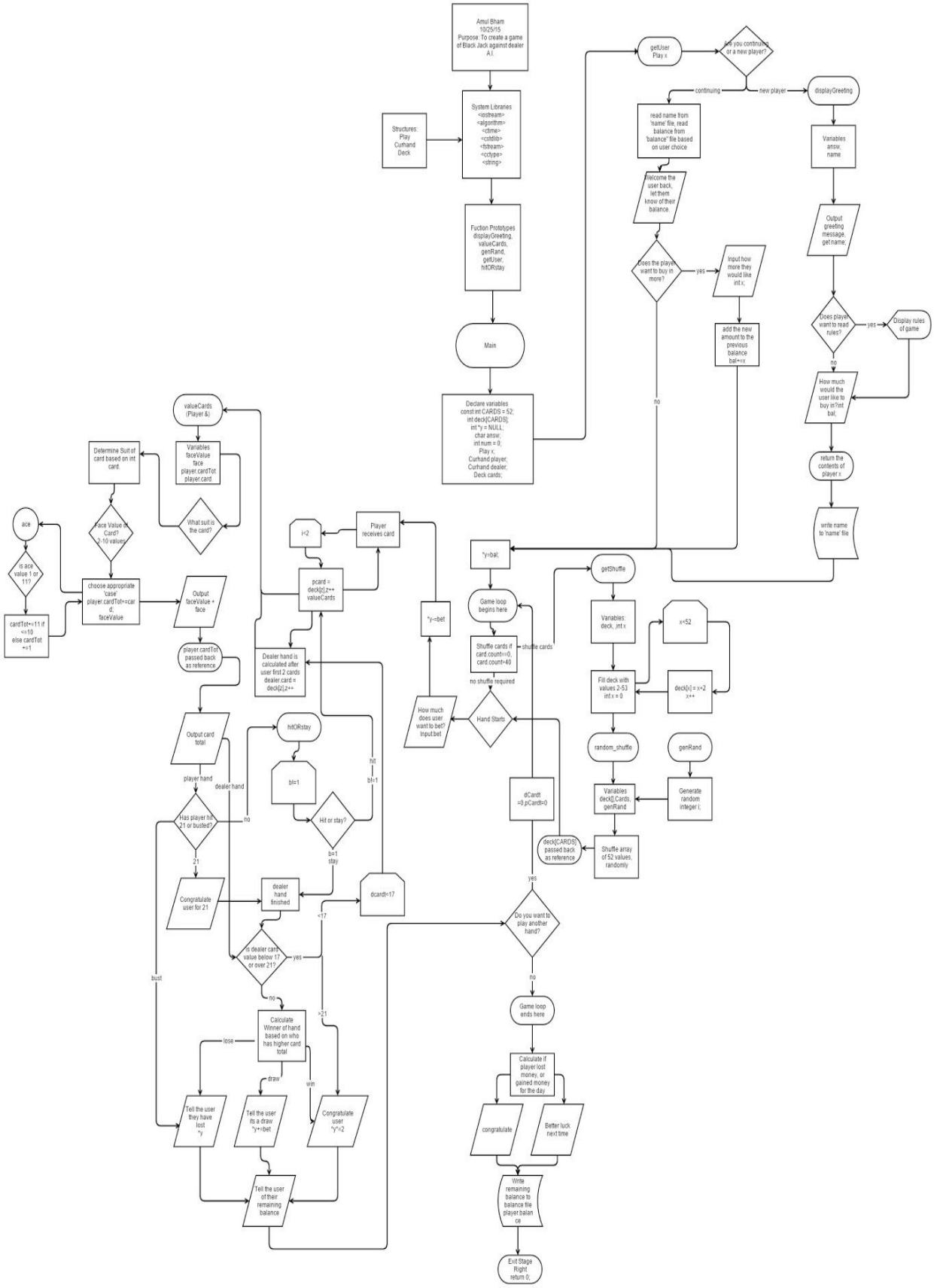
else if the dealer wins or the player goes over 21, the bet is subtracted from the user balance.

User is prompted to play again or walk away

if user walks away, their remaining balance is displayed along with if they came up positive or negative, game ends.

else, game loops to limbo phase, where user must press a key again to start a new hand.

Before program finishes , the remaining balance is saved to a file for the player to continue once they play again.


Amul Bham
10/25/15
Purpose :To create a game of Black Jack against dealer A.I.
Structures: Play Curhand Deck
System Libraries
<iostream>
<algorithm>
<ctime>
<cstdlib>
<fstream>
<cctype>
<string>
Fuction Prototypes displayGreeting, valueCards, genRand, getUser, hitORstay
Main
Declare variables
const int CARDS = 52;
int deck[CARDS];
int *y = NULL;
char answ;
int num = 0;
Play x;
Curhand player;
Curhand dealer;
Deck cards;
getUser Play x
Are you continuing or a new player?
continuing
new player
displayGreeting
read name from 'name' file, read balance from "balance" file based on user choice
Variables answ, name
Welcome the user back, let them know of their balance
Output greeting message, get name;
Does the player want to buy in more?
yes
Input how more they would like int x;
add the new amount to the previous balance
bal+=x
no
Does player want to read rules?
yes
Display rules of game
no
How much would the user like to buy in?:int bal;
return the contents of player x
write name to 'name' file
*y=bal;
getShuffle
Variables: deck, int x
x<52
Fill deck with values 2-53
int x = 0
deck[x] = x+2
x++
random_shuffle
genRand
Variables deck[],Cards, genRand
Generate random integer i;
Shuffle array of 52 values, randomly
deck[CARDS] passed back as reference
shuffle cards
Game loop begins here
Shuffle cards if card.count==0, card.count=40
no shuffle required
Hand Starts
How much does user want to bet? Input:bet
*y-=bet
dCardt =0,pCardt=0
Player receives card
i<2
pcard = deck[z];z++
valueCards
Dealer hand is calculated after user first 2 cards
dealer.card = deck[z];z++
valueCards (Player &)
Variables faceValue face player.cardTot player.card
What suit is the card?
Determine Suit of card based on int card.
Face Value of Card? 2-10 values
ace
Is ace value 1 or 11?
cardTot+=11 if <=10 else cardTot +=1
choose appropriate 'case' player.cardTot+=card, faceValue
Output faceValue + face
player.cardTot passed back as reference
Output card total
player hand
dealer hand
Has player hit 21 or busted?
no
21
bust
hitORstay
bl=1
Hit or stay?
hit
bl=1
b=1 stay
Congratulate user for 21
dealer hand finished
dcardt<17
<17
>21
Is dealer card value below 17 or over 21?
yes
no
Calculate Winner of hand based on who has higher card total
lose
draw
win
Tell the user they have lost *y
Tell the user its a draw *y+=bet
Congratulate user *y*=2
Tell the user of their remaining balance
Do you want to play another hand?
yes
no
Game loop ends here
Calculate if player lost money, or gained money for the day
congratulate
Better luck next time
Write remaining balance to balance file player.balance
Exit Stage Right
return 0;

Major Variables:

Type	Variable Name	Description	Location
Integer	Deck[]	Array that stores the number of cards in a deck 0-52	init() ,getShuffle();
	CARDS	represents the number of cards in a deck	init(),getShuffle();
	*y	takes the value of the players balance at the beginning of the game and changes in throughout the program	init()
	begBal	set = to the original balance, subtracted from y* at the end of the game to determine how much the user has lost, won after all the hands are played.	init()
	b	used to guide program to end loop	hitORstay()
Boolean	win	used to track if the user hits a value of 21 to guide the program to finish the dealer hand and not prompt a hit or stay	init()
Int	num	Used to track location in which the player data should be written to the file	

String/C-string	name[25]	stores the player's name.	init(), displayGreeting(),getUser();
	buyin	used to store player buy in amount and then converted to an integer for to add to the player balance	
	faceValue	returns the integer card value as a string	valueCards()
	face	represents the face of the card	valueCards()
Character	newcard	used to trigger a hit or stay after each card draw	init()
	answ	used to determine if user wants to hear rules, also if user wants to play another hand. Localized in separate functions	init(), displayGreeting()
	c	used to determine if user is returning or is brand new	getUser();
Structure Variables	Curhand player	Structure variable used to track the current hand of the player currently playing	init()

	Curhand dealer	Structure variable used to track the current hand of the dealer	
	Play x	Structure variable of type Play used to track the player information prior to and after the hand is played. includes the name and the balance of the current player	Init(),getUser()
	Deck cards	Used to initialize a deck of cards, as well as keep count of the card in use, as well as increment the cards	init()
Structures	Play	Keeps track of the current player information x.balance, x.name;	Elements: char name[25], int balance
	Curhand	Keeps track of the current hand information for the player and dealer x.cards,x.cardTot. dealer.cardTot, dealer.cards	Elements: int cards, cardTot, bet

	Deck	Stores the values of the deck and tracks which card is being used in the deck cards.count	Elements: int deck[52], int count

C++ Constructs Chapters 9-12:

Chapter	New Concept	Implementation
Chapter 9	pointer variable: I used to set = to the balance, so I could make changes to the pointer versus the actual variable *y	At the beginning of game loop int *y = &x.balance.
	Pointer Arithmetic	After the hand played out I subtracted the bet from the value of the pointer *y -=player.bet; OR *y+=player.bet

Chapter 10	Character Testing:
	I used several character functions to validate user input, allowed me to not worry about if the user entered in uppercase letters
		getUser function:
		if (tolower(c)) == 'c'
		if (tolower(x)) == 'y'

	Character case conversions	getUser, main Converted all user input for simple questions to lowercase using tolower()

	C-strings:
	Used to store the name of the user, as well as general inputs for decision that had to be made in the game.
		structure Play{
		char name [25]
		}

	Allowed for easier input validation	
	Library functions with C-strings Used cin.getline and cin.get to get character input, allowed me to set a limit on the user input so it did not go out of the bounds of the character array	getUser Function, display Greeting functions. cin.get(c) cin.ignore(); cin.getline(name,25)
	C-string numeric conversion functions: used to convert the player balance to a int, allowed the program to account for , and $ and simply convert the numbers in the c-string to an int	getUser function : cin.getline (y,6) -> play.balance+=atoi(y)
	String Objects	Used in the getvalue Function: used to store the suit name, and the value of the suit

Chapter 11	Abstract Data Types Put similar information into structures such as the the current hand, which has the same structure for the dealer and the player, as well as storing player information	Structures located at the start of the program.

	Combining data into structures. Instead of making a separate structure for the dealer and the player hand, I noticed that regardless, they use the same variables with none that are extra and thus could be combined into one structure	Struct Play {} Consists of information about the player and the dealer because they use the same data types and thus do not need to be separated.
	Accessing Member Structures	Used the dot operator throughout my program to access different aspects of the structure for the given variable. For example: When I needed to increment the deck card I used cards.count++ When I needed to change player card or dealer card I could just used player.cards or dealer.cards
	Structures as Function Arguments - made it far easier to change information about all the elements of a given variable as I could freely pass that variable information to functions instead of passing each individual data type.	I passed the structures as arguments so that I could change the information of a variable and all of its elements. In getUser Function, I passed the structure x, which consists of the player name and balance, instead of passing those values individually, I passed by reference to change the values. Inside the getUser function, I then passed the structure to to the getName function if the user was new.

Chapter 12	File operations: Used fstream for reading and writing to files. used ios in, out, binary	Used in the getUser Function fstream player_info player_info.open("player.txt", ios::in | ios::binary);
	Member functions for reading and writing files used the read and write functions for writing and reading from a binary file	Used the getline functions to get read text from the player_info file, also for reading in the binary file and writing to it. read,write functions
	Binary Files - Used as the main form of storing all the player information to a file. Made it easier as I could store the contents of an entire structure as binary data and then read that data back into the program player_info.open("player.txt", ios::in|ios::out|ios::binary);	getUser function () Used to read the structure data from the file which contained the player information. In addition, when the program was finished, I stored the new player information in the form of a structure binary data for future reading and writing
	reinterpret cast : used it to convert the binary data back to characters, as well as converting it back to binary to write	Used in the getUser function as well as the end of the program when I needed to convert the structure information back to binary data for writing.

	Creating Records with structures - used to store the information and keep a record of the players on file. I used a while loop to constantly read out the file structures until it hit the end of the binary file.	getUser function -> allowed the user to pick which user they would like to play on
		while(!player_info.eof()){
		cout<<"Player "<<count<<endl;
		cout<<"Name: "<<play.name<<endl;
		cout<<"Balance: $"<<play.balance<<endl<<endl;
		
		player_info.read(reinterpret_cast<char *>(&play), sizeof(play));
		count++;
		}

	Random Access Files - When the user selected which account they would like to play on, the program then need to locate itself to that point in the binary file, read in the information from that structure, and then only write new information to that structure as the program played out	I used the seekp and seekg member functions player_info.seekg(sizeof(play)*(num-1), ios::beg); to seek to the point in the file where the user information was located based on the user choice, I then read in the information from that structure, and any updates to that account were only made to that part of the binary file thus preserving the the data before it and after it.

Program:

```
/*
 * File:   main.cpp
 * Author: Amul Bham (Project 1)
 * Purpose: To play Blackjack against
 * A.I. implementing methods learned from
 * chapters 9-12.
 * Created on October 17, 2015, 2:42 PM
 */
//System Libraries
#include <iostream>
#include <algorithm>
#include <ctime>
#include <cstdlib>
#include <fstream>
#include <cctype>
#include <string>

using namespace std;
//Structures
//Play - accounts for the player data, to be stored in player_info.txt
struct Play {
   char name[25]; //The user name
   int balance;   //The user balance
};
struct Curhand{
   int cards;  //the players cards for the current hand
   int cardTot;//the player cart total for the current hand
   int bet;    //the player bet for the current hand
};
struct Deck {//allocates an array with 52 values, keeps track of current

   int deck[52];//deck of cards with 52 values
   int count; //The current card number in the deck .

};

//Global Constants

//User Libraries

//Function Prototypes
void displayGreeting(Play &); //function for greeting player
```

```
void valueCards(Curhand &);       //function add card values + suit
int genRand (int i) { srand (time(0)); return rand()%i;} //function to generate a random number
void getShuffle(int deck[],int cards); //creates deck and shuffles cards
void getUser(Play &,int &); //determines user, initiates greeting
void hitORstay(Curhand &,Deck &);//handles logic for hitting/staying

//Execution begins here
int main(int argc, char** argv) {
   //Declare Variables
    const int CARDS = 52; //Number of cards in a standard deck
    int deck[CARDS];//deck of cards with 52 values
    int *y = NULL; //set = to the user balance to calculate their winnings/losings
    char answ;     //sentinel value used to trigger new hand or end game
    int num = 0;   //number used to track where the user is in the file
    Play x;        //created to store/restore a players balance and name
    Curhand player;//Object to player the player hand
    Curhand dealer;//object to track the dealer hand
    Deck cards;
    /*call the get user function to determine if the player is
     continuing or starting a new game*/
    //Get the user information, pass the object of the players, with a number
    //to store the block of memory that user occupies on the file
    getUser(x,num);

    int begBal = x.balance; //tracks what the user started out with
    fstream player_info("player.txt", ios::in | ios::out | ios::binary);
  //game loop begins here
  do{
     //Values that must be reset at the start of each hand
     y = &x.balance; //Pointer to track winning/losses of money
     player.cardTot = 0;   //RESET dealer and player hands at the beginning of each hand
     dealer.cardTot = 0;
     //Only seek to the beginning of the record if the player is continuing,
     //If the player is new, the record must be appended versus rewritten
     if(num!=0){
    player_info.seekg(sizeof(Play)*(num-1), ios::beg);
     }
   //Initialize the deck with values 2-53.
   if (cards.count==0 || cards.count>40){ //only shuffles at beginning or when deck is running low
       getShuffle(deck,CARDS);
       cards.count = 0; //Reset the card count for the new deck
       for (int i =0;i<=CARDS;i++){
           cards.deck[i] =  deck[i];
       }
   }

   //Game begins here by asking how the user would like to bet on first hand
   cout<<"How much would you like to bet?"<<endl;
```

```
cin>>player.bet;
*y -=player.bet; //subtract bet from starting balance
cout<<"Okay sounds good, your remaining balance is $"<<*y<<endl;
cout<<"*****************************************************************\n";
//first two player cards are dealt
cout<<"Your first two cards are.... "<<endl;
for(int i = 0;i<2;i++){
player.cards = cards.deck[cards.count]; cards.count++;
valueCards (player);
}
cout<<x.name<<" your hand equates to "<<player.cardTot<<endl<<endl; //first two card total
cout<<"*****************************************************************\n";

bool win = false; /*value to guide program NOT to display 21 message again if
            user draws a 21 on first two cards*/
if(player.cardTot==21){
   cout<<"Congratulations you got blackjack! ☜(˚▽˚)☞"<<endl;
   cout<<"*****************************************************************\n";
   win = true; //prevents a repeat
}

//Then the dealer draws their first card value
cout<<"My first card is a.... "<<endl;
dealer.cards = cards.deck[cards.count]; cards.count++;
valueCards(dealer);
cout<<"My hand so far equates to "<<dealer.cardTot<<endl; //dealer first card total
cout<<"*****************************************************************\n";

/*Player then must decide to hit or stay, loop continues until
 player busts, hits a 21, or decides to stay */
if (!win){ //only runs if player has not already hit a 21
   hitORstay(player,cards);
}

/*At this point, the program determines if the user has busted or reached
blackjack, if user busts, dealer hand is skipped as the user has lost,
else the dealer hand continues until dealer hits 17 or goes over 21*/
 if(player.cardTot==21 && !win){
   cout<<"Congratulations you got blackjack! ☜(˚▽˚)☞"<<endl;
   cout<<"*****************************************************************\n";
 }else if(player.cardTot>21){
   cout<<"Busted! You lose! ¯|_(ツ)_/¯ "<<endl;
cout<<"*****************************************************************\n";

/*Dealer hand continues until dealer hits 17 or higher or goes over 21,
 only executes if player has not busted*/
}if (player.cardTot <=21){
   cout<<"And now, the rest of my hand..."<<endl;
```

```
      do{ //Loop dealer hand till card total at 17 or higher
        dealer.cards = cards.deck[cards.count]; cards.count++;
        valueCards(dealer);
      }while(dealer.cardTot <17);

   cout<<"My Hand equates to "<<dealer.cardTot<<endl;  //dealer card value total
   cout<<"*****************************************************************\n";

   /*Program then determines who had the higher card value total,
  if user wins, they win twice the amount they bet, if they lose user
  loses their bet, if a tie, bet is returned to the user*/
      if (dealer.cardTot<player.cardTot || dealer.cardTot>21){ //if user value is higher or dealer busts
         cout<<"Congratulations "<<x.name<<
            " you won the hand ☜(˚▽˚)☞ "<<endl;
         *y +=player.bet *2; //user wins 2 * the amount they bet
      }else if (dealer.cardTot>player.cardTot && dealer.cardTot<=21){ //if dealer value is higher
         cout<<"Thank you for playing "<<x.name<< ", but you lose (◔◡◔)"<<endl;
      }else if (dealer.cardTot == player.cardTot){ //if card values are equal
         cout<<"Draw!"<<" I guess that means you technically "
               "aren't a loser..."<<x.name<<endl;
         *y +=player.bet; //if draw, user gets the bet back
      }
    }
   /*Program then lets the user know of their remaining balance to inform them
    on their winnings/losings or if they are the same, and prompts the user
    if they would like to play a new hand*/
    cout<<"*****************************************************************\n";
    cout<<"Your remaining balance is $"<<*y<<endl;

    cout<<"*****************************************************************\n";
    cout<<"Do you want to play another hand, or walk away now?"<<endl;
    cout<<"y/n \n";
    cin>>answ;

    /*set dealer and player card totals back to 0 for new hand*/
    player.bet = 0;

  /*if user would like to play a new hand, program loops back to making a bet
   * ,asks the user for a new bet, and the rest of the program
    proceeds as normal, when card 40 (z=40) is reached, the deck shuffles*/
  }while(tolower(answ) == 'y');

  //Output the results of the game, if the user has lost or gained money
  if (x.balance>=begBal){
  cout<<"*****************************************************************\n";
  cout<<"You winnings for the day are $"<<x.balance - begBal<<"\ngo buy yourself"
        " something nice "<<x.name<<endl;
  }else {
```

```
 cout<<"Looks like you lost $"<<begBal - x.balance<<" today, better luck next time
"<<x.name<<endl;
 }

    //write the remaining balance to the balance folder
    //If the user is a new user, then append their information to the end of the file
    //to create a new account
    if (num == 0){
        player_info.close();
        fstream player_info("player.txt",ios::binary| ios::app);
        player_info.write(reinterpret_cast<char *>(&x),sizeof(x));
    }else{//If the user already has an account, rewrite their account with their
        //New balance
       player_info.seekp((num-1) * sizeof(Play), ios::beg);
       player_info.write(reinterpret_cast<char *>(&x),sizeof(x));
    }
    //Close out the player info file to prevent memory leaks
    player_info.close();

   //Exit stage right
   return 0;
}

/*******************************************************************************
                    getUser
 *******************************************************************************

 Purpose : To determine if the user is returning or is a completely new player,
 * if a new player, greeting function is called, else program reads name and
 * balance from txt file. Get the player name and balance.

 Input : name,balance
 Output: greeting function or a welcome back message

 Return --> void
 *******************************************************************************/

void getUser(Play &play,int &num){
   fstream player_info;
   char c;
   //Determine if the user is returning or if it is their first time
   cout<<"Would you like to continue(c) or start a new game(n)? c/n"<<endl;
   cin.get(c);
   cin.ignore();
   int count = 1;

   /*if continuing, read all the players currently in the file and ask
    the user to select which account they would like to play on */
```

```
if (tolower(c) == 'c'){
   player_info.open("player.txt", ios::in | ios::binary);
   //Open the player file for binary input
   cout<<"The current players on record are ..."<<endl;
   //Static cast the binary data to character and display
   player_info.read(reinterpret_cast<char *>(&play), sizeof(play));
  /*Read out all the players, incremented by byte size, each player
   being stored as an object, until the end of the file is reached*/
   while(!player_info.eof()){
      cout<<"Player "<<count<<endl;
      cout<<"Name: "<<play.name<<endl;
      cout<<"Balance: $"<<play.balance<<endl<<endl;
      player_info.read(reinterpret_cast<char *>(&play), sizeof(play));
      count++;
   }
   //Close the file to prevent memory leaks
    player_info.close();
    //Allow the user to choose an account to play on
    cout<<"Enter the player number of the account you would\n"
          "like to play on"<<endl;
    cin>>num;
    //Read in the player information based on the account choosen
    player_info.open("player.txt", ios::in|ios::out|ios::binary);
    //Seek to the memory location of the user account, based on the num
    player_info.seekg(sizeof(play)*(num-1), ios::beg);
    //Read in the player information, including their name and balance
    player_info.read(reinterpret_cast<char *>(&play),sizeof(play));
    cout<<"Welcome Back "<<play.name<<"!"<<endl;
    //Determine if the user would like to purchase more coins
    cout<<"Your current balance is $"<<play.balance<<", would"
          " you like to buy in more? y/n"<<endl;
    char x; cin>>x;
    if (tolower(x) == 'y'){
       cout<<"Enter how much more you like to buy in or type 0 for none "<<play.name<<endl;
    //Get the buy in as a cstring to prevent run time errors, convert to int using atoi.
    char y[6];cin.ignore();  cin.getline(y,6);
    play.balance+=atoi(y);} //Add the buy in to the total player balance
    cout<<"Your current balance is $"<<play.balance<<endl;
          player_info.close();

/*if the user is new, call the greeting function and input the name
 to the file*/
}else{
//Begin greeting, get user name
displayGreeting(play);
cout<<"*****************************************************************\n";
//Determine how much the player wants to buy into game
cout<<"How much would you like to buy in "<<play.name<<" ?"<<endl;
```

```
   cin>>play.balance;
   }
}


/*************************************************************************
                    displayGreeting
 *************************************************************************

 Purpose : to introduce the user to the game, get the user name to create
 a more personal experience, and then output the rules depending on if the
 user knows how to play or out

 Input : user name, answer
 Output: Rules of the game

 Return --> user name to the main program
 *************************************************************************/

void displayGreeting(Play &play){
   //Declare Variables
   char answ; //sentinel value used to trigger rules or not

   cout<<"               BlackJack ☜(˚▽˚)☞"<<endl;
   cout<<"*****************************************************************\n";
   cout<<"Welcome to the Casino Bham!\nMy name is Amul (◕‿◕) and i will "
         "be your dealer today!\n";
   cout<<"Your name and balance will be saved when you are finished....\n";
   cout<<"What is your name anyway? "<<endl;
   cin.getline(play.name,25); //Get user name, to create a more personal experience
   cout<<"Ah! nice to meet you "<<play.name<<", you have a lovely name";
   cout<<"\nAnyways, Im going to assume you know how to play BlackJack"
         " as you would be foolish\nto play against me without any "
         "experience... "<<endl;
   cout<<"Would you like the rules explained? y/n"<<endl;
   cin>>answ;//Only display rules if user does not know how to play to avoid

   if (tolower(answ) == 'y'){
   //Program explains rules of the game
      cout<<"Well this is awkward... \n anyways ill try to explain my best,\n"
            "and do not expect me to go easy on you just because you are a "
            "rookie... "<<endl;
     cout<<"*****************************************************************\n";
      cout<<"Basically the point of the game is to get to 21 or as close as"
            "possible without going over,\notherwise you lose"
            "\n,in addition, if my card total is higher than yours"
            " you also lose \n";
     cout<<"*****************************************************************\n";
```

```
      cout<<"I will deal 2 cards from a regular deck of cards \n"
            "depending on the total value of these cards, you must\n"
            "decide if you want another card or stay where you are (hit) \n"
            "Also, note that my job as a dealer is to always hit if my card\n"
            "value is below 17,if i bust (go over 21) then you automatically\n"
            "win. So its your choice whether you want to pursue a higher card "
            "value(at the risk of busting)\nor stay and wait for the dealer to bust\n"
            "If we score a tie, then the hand ends in a draw \n";
      cout<<"Also, Jack, Queens, King all have a value of 10\n"
            "while the Ace has a value of either 1 or 11\n"
            "All other cards represent their face value\n";
     cout<<"*********************************************************************\n";
      cout<<"Got it? GOOD! Let me know when you are ready to play"<<endl;
   }else { //program skips to here if user knows the rules
      cout<<"Thank gosh, I hate explaining rules...⊙﹏⊙\n";
   }
}

/*****************************************************************************
                    GetShuffle
 *****************************************************************************
 Purpose : Receive the array that represents a deck of cards, initialize it
 * with numbers 2-54, then shuffle the number around, stimulating
 * the shuffling of a deck of cards

 Input : the array deck[CARDS] - deck of cards

 Return --> the shuffled array back to main
 *****************************************************************************/

void getShuffle(int deck[],int cards){
   srand (time(0));
   for(int x=0;x <cards;x++){
      deck[x]= x+2;
      }
    /*Shuffle the deck for all values, using a random number
    to determine how its shuffled*/
      random_shuffle(&deck[0], &deck[52],genRand);
}
/*****************************************************************************
                    hitORstay
 *****************************************************************************
 Purpose : To determine if the user would like to hit or stay based on their
 * current card value, if hit, loop until user busts, hits 21, or decides to
 * stay

 Input : The card drawn by player, the player card total from previous two cards,
 * the deck, and the card # of the deck.
```

```
 Return --> void
 ********************************************************************************/
void hitORstay(Curhand &player,Deck &cards){
int b =0;      //sentinel value to break out of player card loop
       do{
          char nxtC;  //variable used to determine if user wants to hit or stay

          cout<<"Would you like to "
          "hit or stay? h/s"<<endl;
          cin.ignore();
          cin.get(nxtC);

          if (tolower(nxtC) == 'h'){
          player.cards = cards.deck[cards.count]; cards.count++;
          valueCards (player);
          cout<<player.cardTot<<" is your new card total"<<endl;
          cout<<"*************************************************************\n";

          }if (player.cardTot>21 || player.cardTot == 21 || tolower(nxtC)=='s'){
             b = 1;
     cout<<"*********************************************************************\n";
          }
       }while( b != 1);

}

/*******************************************************************************
                     Get ValueCards
 *******************************************************************************
 Purpose : Receives the integer value of each card, adds them to the card
 * total, and then assigns a string value to the card in addition to a suit

 Input : card total, integer card value
 Output: Card value and suit in the form of a string

 Return --> card total to the main program for further calculation
 ********************************************************************************/

 void valueCards(Curhand &player){
       string faceValue; //face value of card
       string face;      //suit of the card
   /*The suit of the card is determined by the value of the card(2-54),
    then a value is subtracted so the card falls between 2-14*/
          if (player.cards<=14){
             face = "Hearts";
          }else if (player.cards>14 &&player.cards<=27){
             player.cards -= 13;
```

```
      face = "Diamonds";
  }else if (player.cards>27 && player.cards<=40){
      player.cards -= 26;
      face = "Cloves";
  }else if (player.cards>40){
      player.cards -=39;
      face = "Spades";
    }
/*switch statement used to calculate card total, face value is assigned*/
switch(player.cards){
    case 2:
       player.cardTot+=2;
       faceValue="Two";
       break;
    case 3:
       player.cardTot+=3;
       faceValue = "Three";
       break;
    case 4:
       player.cardTot+=4;
       faceValue = "Four";
       break;
    case 5:
       player.cardTot+=5;
       faceValue = "Five";
       break;
    case 6:
       player.cardTot+=6;
       faceValue = "Six";
       break;
    case 7:
       player.cardTot+=7;
        faceValue = "Seven";
       break;
    case 8:
       player.cardTot+=8;
       faceValue = "Eight";
       break;
    case 9:
       player.cardTot+=9;
       faceValue = "Nine";
       break;
    case 10:
       player.cardTot+=10;
       faceValue = "Ten";
       break;
    case 11:
       player.cardTot+=10;
```

```
            faceValue = "Jack";
            break;
         case 12:
            player.cardTot+=10;
            faceValue = "Queen";
            break;
         case 13:
            player.cardTot+=10;
            faceValue = "King";
            break;
         /*since ace represents two values(1 & 11), program must determine
         which to represent, if card total is below or equal to ten, it
         represents an 11 else it represents a 1*/
         case 14:
            faceValue = "Ace";
            if(player.cardTot>10){
              player.cardTot+=1;
            }else if (player.cardTot<=10){
               player.cardTot+=11;}
   }
      player.cards = 0;
      //Output the face value and the suit of the card
      cout<<faceValue<<" of "<<face<<endl;

}
```