Project 1

<Black Jack>

CSC 17A
Section 48591
Due: November 14, 2021
Arlene Sagaoinit

Introduction

Game: Black Jack

Black jack or 21 is the widely known casino card game in which it uses a deck of 52 cards played by one player and one dealer or multiple players and one dealer with the rules being the same. Player is dealt two cards to start and dealer deals themselves two cards. Player can take a look at both their cards and only the dealer's second card is not shown until the player is satisfied with their cards.

The objective of the game is to obtain a higher score than the dealer without going over the sum 21. The cards 2-10 hold the value of their face, J,Q, and K hold the value 10, and the A card can either hold the value 1 or 11. If the player is not satisfied with the sum of the first two cards dealt to them, they can "hit" or ask for another card when it is their turn. As long as the sum of their cards is less than 21, they are still in the game. A rule for the dealer is that they have to hit if their sum is less than 17; if it is 17 or greater they have to stay.

If the player's sum is greater than 21, they automatically lose without waiting for the dealer. If player's sum is less than 21 but not greater than the dealers, they lose. If the player's sum is less than 21 and greater than dealer's sum, they win. If the player's sum is 21 and the dealer's sum is not 21, they win.

Summary

Project Size: 415 Lines of code
Number of variables: 35

I have a total of 8 different versions of the game, each one with a small amount of modifications. I worked on it within a span of about three weeks, 3-4 days a week, about 3-4 hours a day. So around a total of 35-40 hours.

Concepts from ch 9-12 used

- Arrays, Pointers
- Allocated memory
- Structures
- Nested structures
- Array of structures
- String member functions
- Enumerators
- Binary files, member functions

I ran into a lot of difficulties, starting with how to implement the rule for the ace card. It didn't seem like a good idea to ask the user to set the value to either a 1 or 11 every time an ace card is pulled since they can change it in any time of the round. So I thought I would have to constantly ask the user if they want to keep the value the same, which made it ugly. So instead I defaulted the value to 11 until the user gets another card and goes over, I record the index of the ace card pulled to then change the value to 1 and just prompt for a hit or stay. Spent like 3 days on that.

Another difficulty was that I wanted to have the user's hand information inside the structure (array of hand, hand sum, size) reset or deleted without the other stats in the structure being reset. So that's when I created a nested structure in which I can delete the inner structure within the do-while loop, to have the hand contents initialized back to zero with every round/loop.

```
struct Player{
    Hand *plyrHnd;
    int betAmt;
    int chipsWon;
    int totChips;
    unsigned int games;
    unsigned int wins;
    unsigned int losses;
    bool status;
};
```

```
struct Hand{
    int *cards;
    int index;
    int handTot;
    int aceElem;
};
```

delete

Another was writing multiple records to a binary file. I wanted to write player's stats of each round into a binary file then output all of them but each record read contained the same stats. Ex: three games played, first bet 50, second bet 20, third bet 30, but each record had only the last game's stats.

```
****************************************
Game #: 3
Bet Amount: 30
Chips Won: 60
****************************************
Game #: 3
Bet Amount: 30
Chips Won: 60
****************************************
Game #: 3
Bet Amount: 30
Chips Won: 60
****************************************
```

It only took the last game's stats

Then I went to the lab and realized I wrote the .write() function wrong and changed the sizeof argument to the size of the struct instead of the size of the object from the struct. I still don't really understand how that made it work.

Description

How I went about creating the program (each is a new version made):

v.1) Created a single gameplay, no rules implemented yet, had a char array holding face cards to pull from and store in another

v.2) Placed simple actions into their own functions

v.3) Created a hit function, ace function, structure to hold player's card Information (includes: pointer/array, int variables), included dealer's hit rule of hitting before 17 and stay after 17

v.4) Created nested structure to separate player's card info and player's game stats, and added loop for multiple game plays, create and delete player's card info inside loop

v.5) Added another structure to hold the info of a single face card (face, suit, value), and an array of structures to represent deck[52] instead of a char array with just faces. Pull from faces[], suits[], cardVals[], store into deck[] using indexes

v.6) Write a single record to a binary file, and read from it

v.7) Implemented an enumerator and cstring

v.8) Wrote multiple records to a binary file by writing per loop, read from them and placed structures in their own .h files

Sample Input/Ouput

For bet amount: input value between 20-100, will keep prompting until correct value inputted.

Prompts: only takes ‘y’,’Y’,’n’,’N’, will keep prompting until either of those inputted.

```
--------------------------------------
Place your bet: 50
```

Input 20-100

```
--------------------------------------
Place your bet: 50
Your cards: 9D KC
Total: 19

Dealers first card: 6H
Hit? (y/n):
```

Type ‘y’ or ‘n’

```
--------------------------------------
Place your bet: 50
Your cards: 9D KC
Total: 19

Dealers first card: 6H
Hit? (y/n): n
Total: 19

Dealers second card: 3S
Dealer hits: 9C
Dealers total: 18

You win!
Chips Won: 50
Would you like to play again? (y/n)
```

Type ‘y’ or ‘n’

```
****************************************
Game #: 1
Status: Win
Bet Amount: 50
Chips Won: 50
****************************************
        Total Games Played:     1
        Total Wins:             1
        Total Losses:           0
        Overall chips won:     50
```

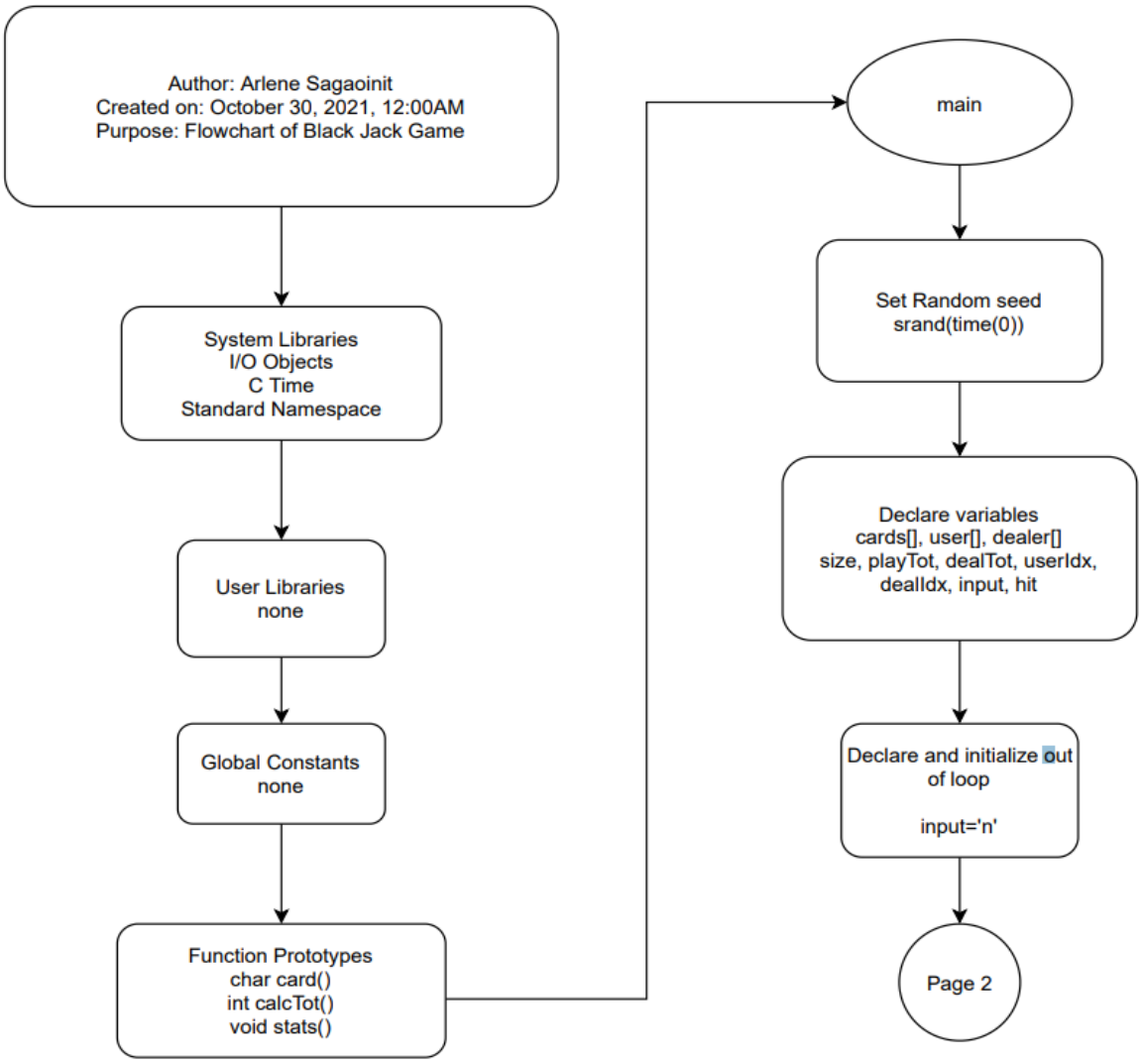
Pseudocode

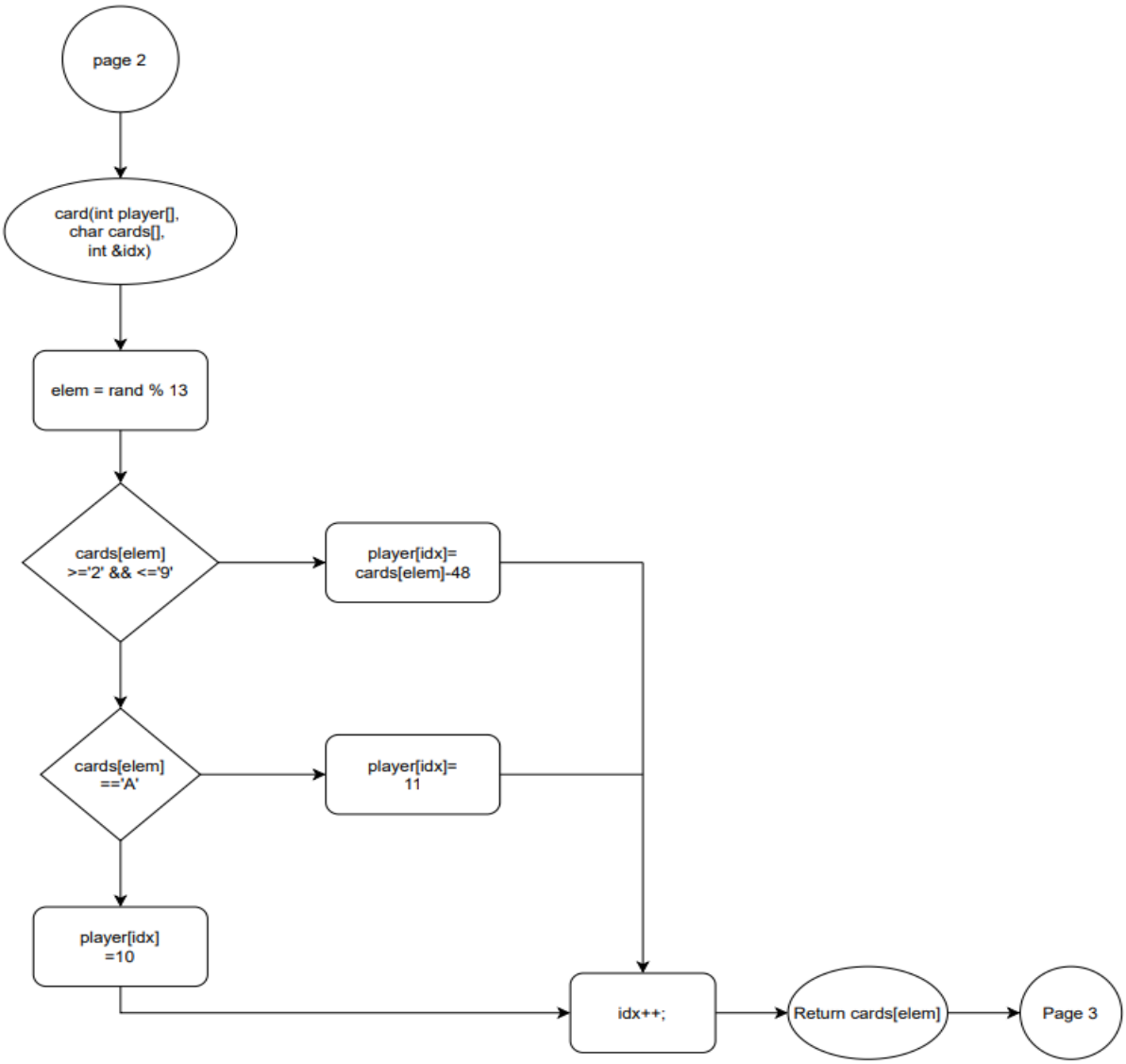
```
//Include libraries: iostream, ctime
//Create random number generator srand(static_cast<unsigned int>(time(0)))
//Create char array holding face cards
//char cards[]={'2','3','4','5','6','7','8','9','T','J','Q','K','A'}

//Initialize values
//Create player and dealers hand arrays
// set size of hand to 10
// set playIdx of hand to 0
// set dealIdx of hand to 0
// int player[size]
// int dealer[size]

//Grab a face card from cards[] place in player[playIdx]
// if card between '2' and '9'
//       convert to int cards[rand()%13]-48;
//       set player[playIdx]=cards[rand()%13]-48;
// else card is either 'T','J','Q','K','A'
//       if card is 'A'
//       set player[playIdx]=11;
//       else card is 'T','J','Q','K'
```

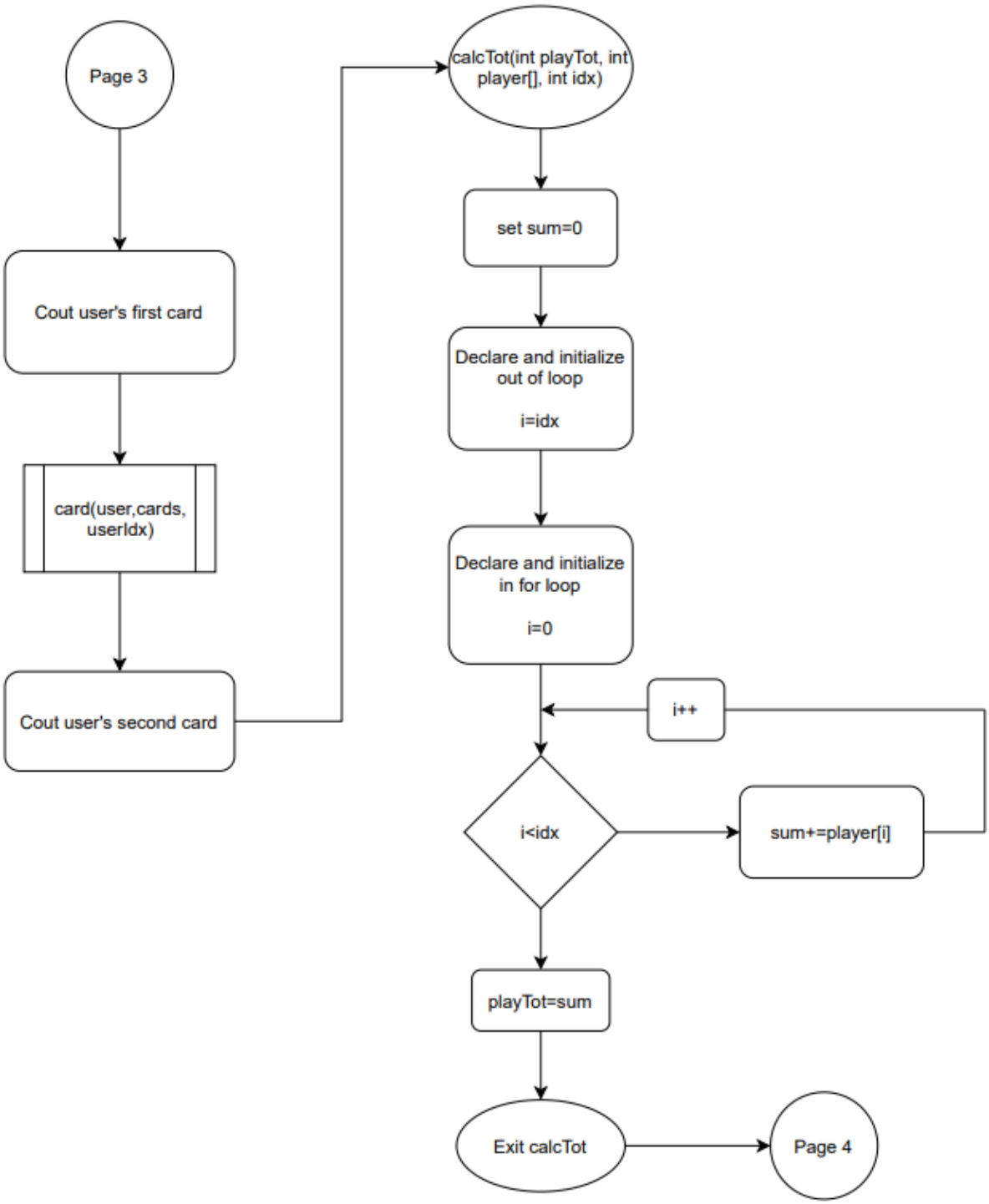
```
//      set player[playIdx]=10;
// increment playIdx

//Assign a second card to player (repeat above step)

//Assign a card to dealer
//Grab a face card from cards[] place in dealer[dealIdx]
// if card between '2' and '9'
//      convert to int cards[rand()%13]-48;
//      set dealer[dealIdx]=cards[rand()%13]-48;
// else card is either 'T','J','Q','K','A'
//      if card is 'A'
//      set dealer[dealIdx]=11;
//      else card is 'T','J','Q','K'
//      set dealer[dealIdx]=10;
// increment dealIdx

//Map outputs
//Calculate total of player and dealers hand
// set playTot to 0
// loop from index 0 to playIdx, index++
//      Add player[index] to playTot
// set dealTot to 0
// loop from index 0 to dealIdx, index++
//      Add dealer[index] to dealTot

//If player total < 21 prompt player to hit or stay
// set bool quit to false
// define char hit;
// while playTot < 21 && quit is false
//      cout "Hit?"
//      Assign input to hit variable
//      if hit == 'y'
//      assign card to player[playIdx]
//      calculate total
```

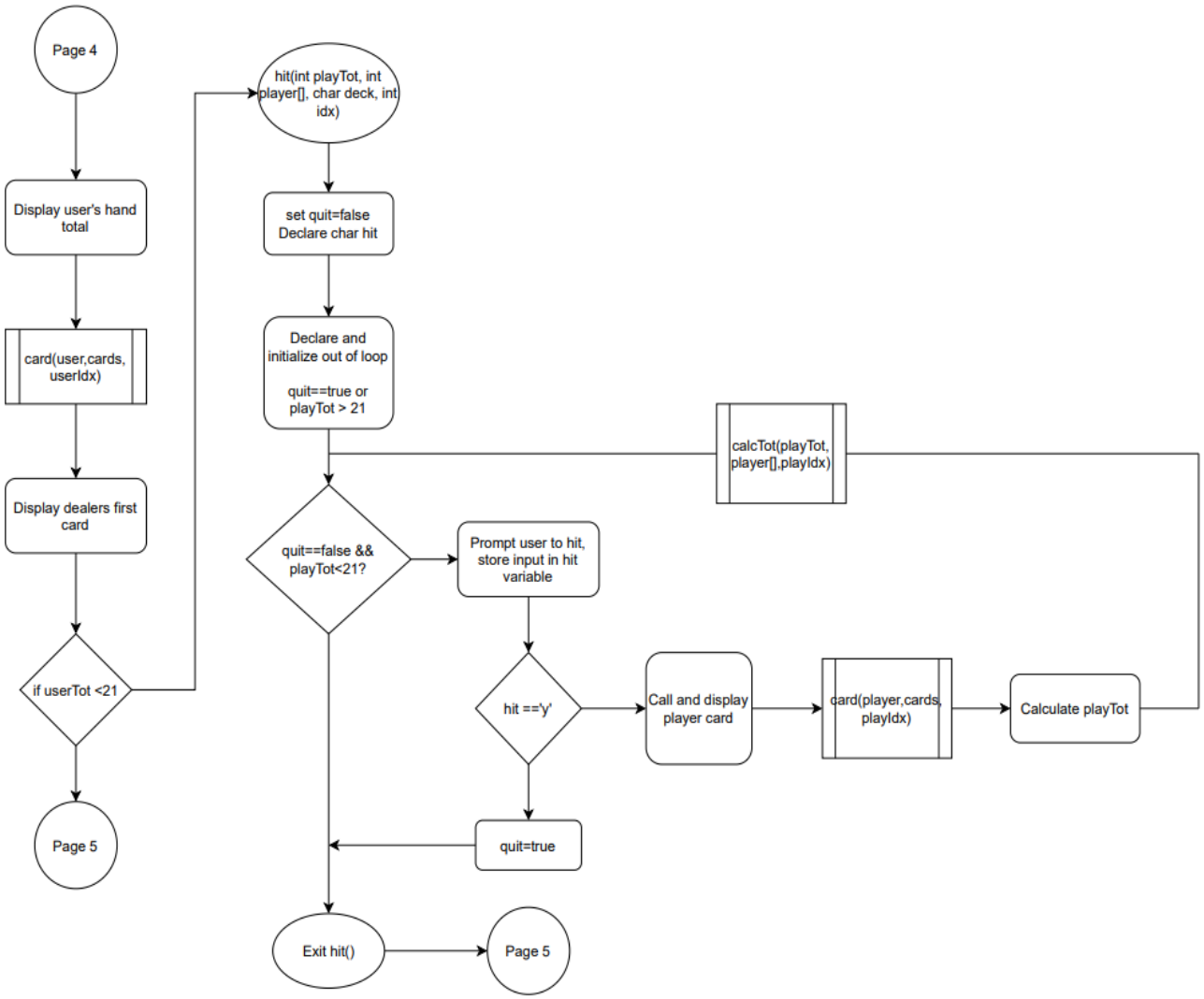
```
//       else hit == 'n'
//       set quit to true

//Display results
//Compare totals and display win or loss
// if playTot > 21
//       cout "loser"
// else if playTot < 21
//       if dealTot < 21
//       if playTot > dealTot
//                cout "winner"
//       else playTot < or equal to dealTot
//                cout "loser"
//       else if  dealTot > 21
//       cout "winner"
//       else dealTot==21
//       cout "loser"
//else playTot == 21
//       if dealTot!= 21
//       cout "Black jack!"
//       else dealTot =21
//       cout "Tie! Loser"
```

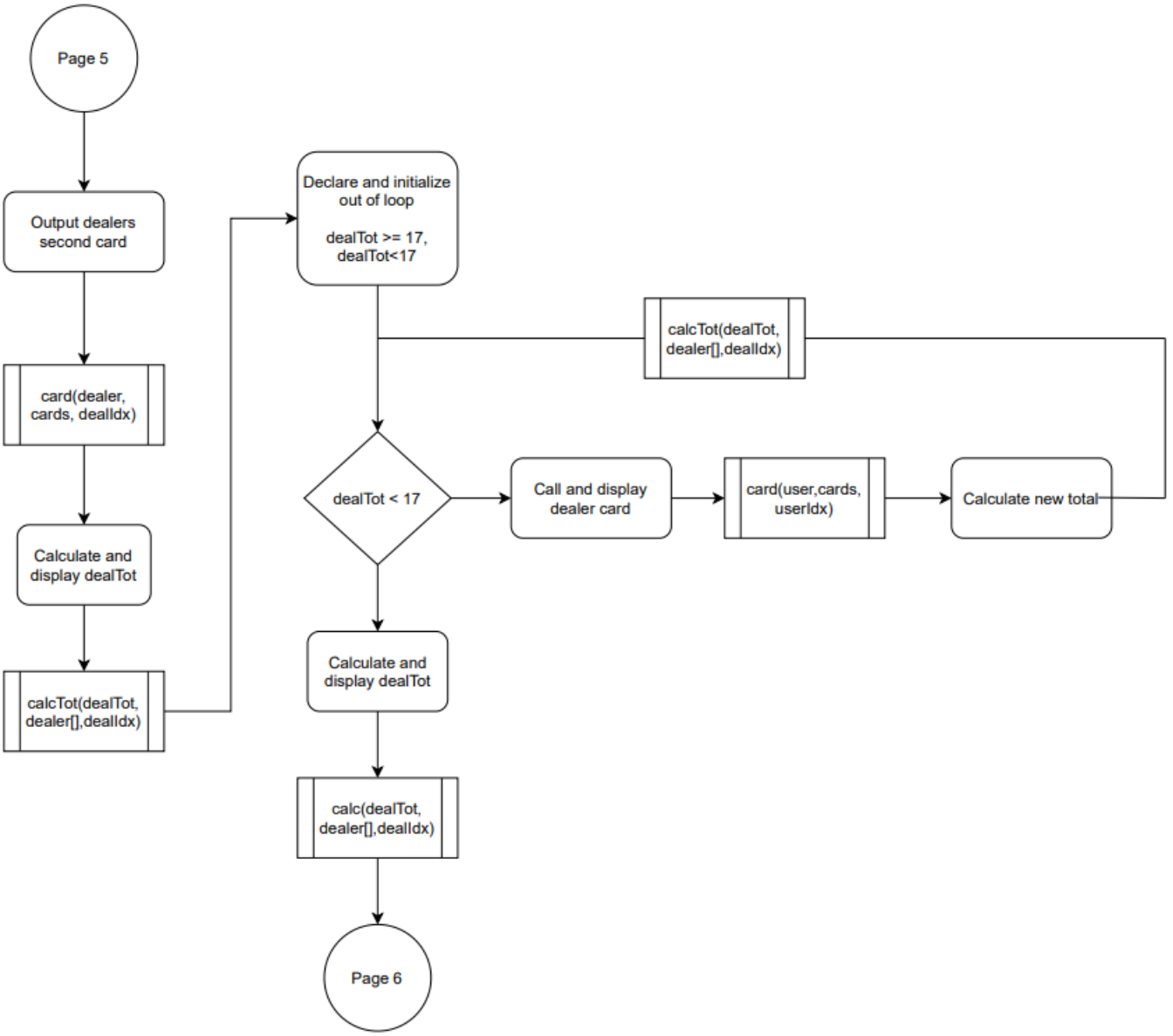
FlowChart

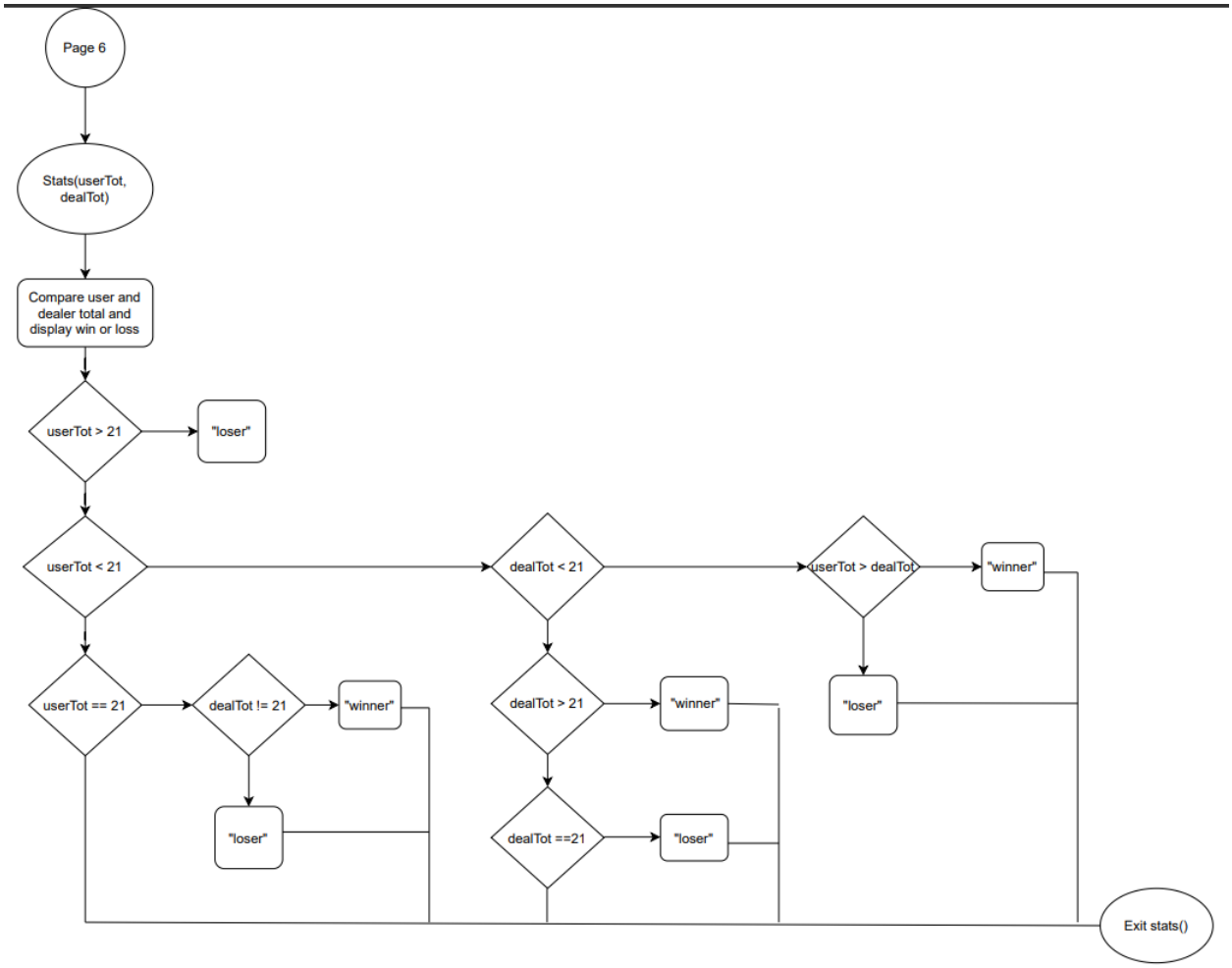

Author: Arlene Sagaoinit
Created on: October 30, 2021, 12:00AM
Purpose: Flowchart of Black Jack Game
System Libraries
I/O Objects
C Time
Standard Namespace
User Libraries
none
Global Constants
none
Function Prototypes
char card()
int calcTot()
void stats()
main
Set Random seed
srand(time(0))
Declare variables
cards[], user[], dealer[]
size, playTot, dealTot, userIdx,
dealIdx, input, hit
Declare and initialize out
of loop
input='n'
Page 2


page 2
card(int player[],
char cards[],
int &idx)
elem = rand % 13
cards[elem]
>='2' && <='9'
player[idx]=
cards[elem]-48
cards[elem]
=='A'
player[idx]=
11
player[idx]
=10
idx++;
Return cards[elem]
Page 3


Page 3
Cout user's first card
card(user,cards, userIdx)
Cout user's second card
calcTot(int playTot, int player[], int idx)
set sum=0
Declare and initialize out of loop
i=idx
Declare and initialize in for loop
i=0
i++
i<idx
sum+=player[i]
playTot=sum
Exit calcTot
Page 4


Page 4
Display user's hand total
card(user,cards, userIdx)
Display dealers first card
if userTot <21
Page 5
hit(int playTot, int player[], char deck, int idx)
set quit=false Declare char hit
Declare and initialize out of loop
quit==true or playTot > 21
calcTot(playTot, player[],playIdx)
quit==false && playTot<21?
Prompt user to hit, store input in hit variable
hit =='y'
Call and display player card
card(player,cards, playIdx)
Calculate playTot
quit=true
Exit hit()
Page 5


Page 5
Output dealers second card
card(dealer, cards, dealIdx)
Calculate and display dealTot
calcTot(dealTot, dealer[],dealIdx)
Declare and initialize out of loop
dealTot >= 17, dealTot<17
calcTot(dealTot, dealer[],dealIdx)
dealTot < 17
Call and display dealer card
card(user,cards, userIdx)
Calculate new total
Calculate and display dealTot
calc(dealTot, dealer[],dealIdx)
Page 6



Major Variables

Variable	Description	Location
Player *user	Pointer to struct holding users cards, stats info	main() Line 65
Player *dealer	Pointer to struct holding dealer info	main() Line 66
string faces[]	Static array holding the face cards	main() Line 68
char suits[]	C-string holding suits	main()

		Line70
Int cardVals[]	Static array holding the values of each face card in order	main() Line 72
Card deck[52]	Array of structures containing all 52 cards with their face, suit, and value	main() Line 73
user->plyrHnd	Nested Hand structure contains player's hand	main() Line 90
dealer->plyrHnd	Nested Hand structure contains dealer's hand	main() Line 91
user->plyrHnd->handTot	Player's hand sum, updated with each card	main() Line 107
user->plyrHnd->index	Holds the current size of player's hand, updates with each card	string card() Line 371
user->plyrHnd->aceElem	Holds index of ace card pulled	String card() Line 370
Face faceCrd	Enumerator for face cards, implemented in for loop	void crDeck() Line 226
Fstream binFile	Binary file	main() 62
Int index	Holds index in deck[], moves to next index to pull next card after shuffling	string card() Line 372
Player *record	Allocated pointer to struct to hold a record read in from binary file	Player *readBin(fstream &) Line 398

Concepts

Enumerator	enum Face {TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE}; Face faceCrd; for(int faceCrd=TWO; faceCrd<=ACE; faceCrd=static_cast<Face>(faceCrd+1))	Line 33 main(), 55 crDeck(), 226
Structures	struct Player{} struct Hand{} struct Card{}	Player.h Hand.h Card.h
Pointer to struct	Player *user=playr(); Player *dealer=playr();	main() 65,66
Allocated memory	Hand *hand=new Hand; Player *playr=new Player; delete [] user->plyrHnd->cards; delete user->plyrHnd; delete user;	hand(), 197 playr(), 209 main(), 134 main(), 135 main(), 187
Arrays	Int *cards; string faces[]= {"2","3","4","5","6","7","	Hand.h, 16 main(), 68

	8","9","10","J","K","Q","A"}; char suits[]={"DSHC"}; int cardVals[]={2,3,4,5,6,7,8,9,10,10,10,10,11};	 main(), 70 main(), 72
Array of structures	Card deck[52]	main(), 73
C-string	char suits[]={"DSHC"};	main(), 70
Strings	string faces[]={"2","3","4","5","6","7","8","9","10","J","K","Q","A"};	main(), 68
C-string member functions	for(int suit=0; suit<strlen(suits);suit++)	crDeck(), 225
Binary files	fstream binFile;	main(), 60

Binary files member functions	binFile.open("stats.bin",ios::out|ios::binary);
	
	bin.write(reinterpret_cast<char *>(plyr),sizeof(Player));
	
	bin.read(reinterpret_cast<char *>(record),sizeof(Player));
	
	binFile.close();
	
	binFile.eof()
		main(), 63
		
		wrtBin(), 393
		
		readBin(), 399
		
		main(), 159
		
		main(), 171

Records in file	Player *recrd;	main(), 165

	recrd=readBin(binFile);
	print(recrd);
	
	//Loop
	while(!binFile.eof())
	//Reads records then prints to screen
		main(), 168
		main(), 172

Reference

1. Gaddis textbook
2. Lectures
3. Creating a deck of cards
   https://www.youtube.com/watch?v=iMSMVqTlvjM
4. Shuffling an array
   https://www.youtube.com/watch?v=xH3VbMPFFec
5. Shuffling a deck of cards
   https://www.youtube.com/watch?v=py7nc1Pjgto

Program

```
/*
 * File:   main.cpp
 * Author: Arlene Sagaoinit
 * Created on November 12, 2021, 8:00 PM
 * Purpose:  Final version of black jack game
 *               -Placed structures in own .h files
 *               -Added writing multiple records to binary file
 *               -Added bool variable in Player struct for win or loss status
 *               -Added total chips variable in Player struct for overall chips won
 *               -Implemented more functions, copy/paste
 *               -crDeck() - creates deck of 52
```

```
 *              -print() - prints a record from binary file
 *              -endStats() - prints overall gameplay stats at end
 */

//System Libraries
#include <iostream>   //Input/Output Library
#include <ctime>      //srand()
#include <iomanip>    //setw()
#include <cstring>    //strlen()
#include <fstream>    //read/write to file
using namespace std;  //STD Name-space where Library is compiled

//User Libraries
#include "Card.h"     //Struct holding data of single card from deck
#include "Hand.h"     //Struct holding players hand info
#include "Player.h"   //Struct holding players gameplay stats

//Global Constants not Variables
//Math/Physics/Science/Conversions/Dimensions

//Enumerator to hold faces
enum Face {TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE,
TEN, JACK, QUEEN, KING, ACE};

//Function Prototypes
Player *playr();          //Returns a Player struct with initialized values
Hand *hand(int);          //Returns a Hand struct with initialized values
void crDeck(Card [],string [], char [], int []);    //Create deck of 52
string card(Player *, Card [], int &);  //Assigns card to player
void calcTot(Player *);          //Calculate total of hand
void hit(Player *, Card [], int &);    //Ask user if want another card
void stats(Player *, Player *);  //Display win or loss
void chckAce(Player *);       //Check if players hand contains an ace
void shuffle(Card []);        //Passes in deck and shuffles values in deck
void wrtBin(fstream &,Player *); //Write to binary file
```

```
Player *readBin(fstream &);  //Read from binary file
void print(Player *);        //Print records read from binary file
void endStats(Player *);     //Print overall gameplay stats at the end

//Code Begins Execution Here with function main
int main(int argc, char** argv) {
        //Set random number seed once here
        srand(static_cast<unsigned int>(time(0)));

        //Declare variables here
        int size;               //Size of player's hand
        int index;              //Index for deck of cards
        char input;             //User input to play game again (y/n)
        Face faceCrd;           //Enumerator, face cards
        fstream binFile;        //Binary file

        //Initialize variables here
        binFile.open("stats.bin",ios::out|ios::binary);
        size=10;                //Initialize hand size to 10, too big
        Player *user=playr();   //Create a player struct for user
        Player *dealer=playr();         //Create a player struct for dealer
        //Static array to hold face of each card
        string faces[]={"2","3","4","5","6","7","8","9","10","J","K","Q","A"};
        //C string to hold suit of each card
        char suits[]={"DSHC"};  //Diamond, Spade, Heart, Club
        //Static array to hold corresponding face value
        int cardVals[]={2,3,4,5,6,7,8,9,10,10,10,10,11};
        Card deck[52];          //Define an array of structures, each element a
Card struct

        //Create deck of cards
        crDeck(deck,faces,suits,cardVals);

        //Map inputs to outputs here, i.e. the process
        cout<<"Play Blackjack!"<<endl;
```

```
        cout<<"Beat the dealer by gaining a hand total close to 21 without
going over. "<<endl
        <<"If you have an ace card, its value is 11 until you go over 21. When
"
        <<"you go over, your ace will change to 1 and you can continue
hitting "
        <<"until you go over again."<<endl
        <<"Min bet: $20"<<endl<<"Max bet: $500"<<endl;
        do{
        //When end of deck reached (all 52 cards used),reset index back to 0
        index=index>51?0:index;

        cout<<"-------------------------------------"<<endl;
        user->plyrHnd=hand(size);    //Allocate memory to hand info struct
        dealer->plyrHnd=hand(size);
        cout<<"Place your bet: ";
        cin>>user->betAmt;              //Collect users bet amount
        //Error check, amount cant be less than 20 or greater than 100
        while(user->betAmt<20 || user->betAmt>100){
        cout<<"Invalid bet amount"<<endl;
        cout<<"Min: $20 "<<"Max: $100"<<endl;
        cin>>user->betAmt;
        }

        shuffle(deck);              //Shuffle the deck
        //Pull cards and display them
        cout<<"Your cards: "<<card(user,deck,index)<<"
"<<card(user,deck,index)<<endl;
        calcTot(user);              //Calculate total of hand
        chckAce(user);              //Check if ace card pulled, if went over change
val to 1
        //Display users total
        cout<<"Total: "<<user->plyrHnd->handTot<<endl<<endl;

        cout<<"Dealers first card: "<<card(dealer,deck,index)<<endl;
```

```
        //If users hand total < 21, call function to prompt user to hit
        //prompt user while total<21 and until they want to stop
        if(user->plyrHnd->handTot<21){
        hit(user,deck,index);
        cout<<"Total: "<<user->plyrHnd->handTot<<endl<<endl;
        }

        //Grab second card for dealer
        cout<<"Dealers second card: "<<card(dealer,deck,index)<<endl;
        calcTot(dealer);            //Calculate the dealers hand total
        while(dealer->plyrHnd->handTot<17){//Dealer pulls cards until the
total is 17 or greater
        cout<<"Dealer hits: "<<card(dealer,deck,index)<<endl;
        calcTot(dealer);            //Calculate new total
        chckAce(dealer);            //Check if card pulled is an ace
        }
        //Display dealers total
        cout<<"Dealers total: "<<dealer->plyrHnd->handTot<<endl<<endl;

        //Display the results
        stats(user, dealer);        //Compare user and dealer totals and display
win or loss

        //Deallocate memory
        //Delete only the player's hand info inside do while loop to reset their
hand
        //but keep players stats -> betAmt, chipTot, num games, wins, losses
        delete []user->plyrHnd->cards;      //Destroy users hand, hand total,
and size
        delete user->plyrHnd;           //Destroy Hand structure
        delete []dealer->plyrHnd->cards;
        delete dealer->plyrHnd;

        //Write to binary file
        wrtBin(binFile, user);
```

```
//Reset bet amount and chips won for next round
user->betAmt=0;
user->chipsWon=0;

//Prompt user for next game or to end game
cout<<"Would you like to play again? (y/n) ";
cin>>input;
//If input uppercase, convert to lowercase
//User can only enter 'y' or 'n', loop while input invalid
while(tolower(input)!='y'&&tolower(input)!='n'){
cout<<"Invalid please enter 'y' or 'n"<<endl;
cin>>input;
}
}while(tolower(input)=='y');    //Start game again if input == 'y'
cout<<endl;

//Close file
binFile.close();

//End of game, display records of each round played
//Open file again for input
//If file open successful, read from beginning of file
binFile.open("stats.bin",ios::in|ios::binary);
Player *recrd;
if(binFile){
//Read first record
recrd=readBin(binFile);
cout<<"****************************************"<<endl;
//While end of file not reached, read next record and print
while(!binFile.eof()){
print(recrd);
cout<<"****************************************"<<endl;
recrd=readBin(binFile);
}
```

```
        binFile.close();    //Close file
        }else{
        cout<<"Error opening file"<<endl;
        }

        //Print overall gameplay stats
        endStats(user);

        //Deallocate memory
        //Now when user is done playing
        //delete struct containing players stats -> chipTot, num games, wins,
losses
        delete user;
        delete dealer;
        delete recrd;

        return 0;
}

//Create Hand struct containing player's cards info
Hand *hand(int n){
        n=n<2?2:n;              //Initialize size of player's hand
        Hand *hand=new Hand;        //Define struct
        hand->cards=new int[n];  //Define cards array
        hand->index=0;          //Initialize index referenced to hand array
        hand->handTot=0;            //Initalize hand total
        hand->aceElem=-1;
        //Set ace element to -1 until ace card pulled
        //when pulled, set aceElem to ace card's index position in cards array
        return hand;            //Return hand struct
}

//Create a Player struct containing players game stats
Player *playr(){
        Player *playr=new Player;   //Define Player struct
```

```
        playr->betAmt=0;        //Initalize all members to 0
        playr->chipsWon=0;
        playr->games=0;
        playr->wins=0;
        playr->losses=0;
        playr->status=false;    //Win status default to false
        return playr;           //Return Player struct
}

//Create a deck of 52 cards by pulling from faces[], suits[], cardval[]
//Store into deck of Card structures
void crDeck(Card deck[],string faces[], char suits[], int cardVals[]){
        //Use outer for loop to represent suits (0-4)
        //Inner loop for assigning card's face, value, and suit
        int k=0;//Index of
        for(int suit=0; suit<strlen(suits);suit++){
        for(int
faceCrd=TWO;faceCrd<=ACE;faceCrd=static_cast<Face>(faceCrd+1)){
        deck[k].face=faces[faceCrd];  //Assign a face to card
        deck[k].suit=suits[suit];         //Assign a suit
        deck[k].val=cardVals[faceCrd];//Assign the face value
        k++;                        //Increment, k goes to 52 (size of deck array)
        }
        }
}

//Updates players stats and displays a win or loss to user
void stats(Player *playr, Player *dealr){
        if(playr->plyrHnd->handTot>21){ //user goes over, display loss
        //Subtract bet amount from total chips won
        //totChips can't be negative, set to 0 if totChips-betAmt < 0
        if(playr->totChips-playr->betAmt<0)playr->totChips=0;
        else playr->totChips-=playr->betAmt;
        playr->games++;         //Increment games played
        playr->losses++; //Increment losses
```

```
playr->status=false;      //Status false for loss

//Display stats
cout<<"You Lose!"<<endl;
cout<<"Chips Won: "<<playr->chipsWon<<endl;
}else if(playr->plyrHnd->handTot<21){
//If user total and dealer total < 21
if(dealr->plyrHnd->handTot<21){
//and user total > dealer total, display win
if(playr->plyrHnd->handTot>dealr->plyrHnd->handTot){
        playr->totChips+=playr->betAmt;  //Add bet amount to chip total
        playr->chipsWon+=playr->betAmt;
        playr->games++;
        playr->wins++;            //Incrememnt wins
        playr->status=true;       //True for win

        cout<<"You win!"<<endl;
        cout<<"Chips Won: "<<playr->chipsWon<<endl;
//user total < dealer total, display loss
}else{
        if(playr->totChips-playr->betAmt<0)playr->totChips=0;
        else playr->totChips-=playr->betAmt;
        playr->games++;
        playr->losses++;
        playr->status=false;

        cout<<"You Lose!"<<endl;
        cout<<"Chips Won: "<<playr->chipsWon<<endl;
}
//user total < 21 and dealer total > 21, display win
}else if(dealr->plyrHnd->handTot>21){
playr->totChips+=playr->betAmt;
playr->chipsWon+=playr->betAmt;
playr->games++;
playr->wins++;
```

```
playr->status=true;

cout<<"You Win!"<<endl;
cout<<"Chips Won: "<<playr->chipsWon<<endl;
//user total < 21 and dealer total = 21, display loss
}else{
if(playr->totChips-playr->betAmt<0)playr->totChips=0;
else playr->totChips-=playr->betAmt;
playr->games++;
playr->losses++;
playr->status=false;

cout<<"You Lose!"<<endl;
cout<<"Chips Won: "<<playr->chipsWon<<endl;
}
//User total ==21
}else{
//If dealer total != 21, display win
if(playr->plyrHnd->handTot!=dealr->plyrHnd->handTot){
playr->totChips+=playr->betAmt;
playr->chipsWon+=playr->betAmt;
playr->games++;
playr->wins++;
playr->status=true;

cout<<"You Win!"<<endl;
cout<<"Chips Won: "<<playr->chipsWon<<endl;
//Dealer total = 21, display loss
}else{
if(playr->totChips-playr->betAmt<0)playr->totChips=0;
else playr->totChips-=playr->betAmt;
playr->games++;
playr->losses++;
playr->status=false;
```

```
        cout<<"Tie You Lose"<<endl;
        cout<<"Chips Won: "<<playr->chipsWon<<endl;
        }
        }
}

//Function to allow player to hit while hand total < 21
void hit(Player *playr, Card deck[], int &idx){
        bool quit=false;    //Set to true when user wants to stop hitting
        //Prompt user to hit while quit is false and hand total < 21
        while(quit==false&&playr->plyrHnd->handTot<21){
        cout<<"Hit? (y/n): ";
        char hit;
        cin>>hit;
        //If input is uppercase, convert to lowercase
        //Only two options: 'y' or 'n' if not chosen, prompt until one of the two
chosen
        while(tolower(hit)!='y'&&tolower(hit)!='n'){
        cout<<"Invalid please enter 'y' or 'n'"<<endl;
        cin>>hit;
        }
        if(tolower(hit)=='y'){  //If yes
        cout<<"card: "<<card(playr,deck,idx)<<endl; //Pull card
        calcTot(playr);     //Calculate total
        chckAce(playr);   //Check if card pulled is an ace
        }else{
        quit=true;            //If no, set quit to true and exit loop
        }
        }
}

//Function to check players hand for an ace card (aceElem!=-1)
//If hand total > 21, change the value of ace card to 1
void chckAce(Player *playr){
        //If player has an ace and goes over, set value to 1
```

```
        if(playr->plyrHnd->handTot>21&&playr->plyrHnd->aceElem!=-1){
        playr->plyrHnd->cards[playr->plyrHnd->aceElem]=1;
        calcTot(playr);//Recalculate total with ace as 1
        }
}

//Calculate total of players hand
void calcTot(Player *playr){
        int sum=0;          //Variable to hold sum of players hand
        for(int i=0; i<playr->plyrHnd->index;i++)   //Loop through hand
        sum+=playr->plyrHnd->cards[i];              //Add each value in hand to
sum
        playr->plyrHnd->handTot=sum;                //Return sum
}

//Modified to return string instead of char
//Create a random number between 0-51 to represent an element in card
deck
//Store the value of face card from deck into players hand
string card(Player *playr, Card deck[], int &index){
        int elem=index;             //Element number from 0 to 12
        //Store the value of face card from deck into players hand
        playr->plyrHnd->cards[playr->plyrHnd->index]=deck[elem].val;
        //If card selected is an ace, record its index in hand array

if(deck[elem].face=="A")playr->plyrHnd->aceElem=playr->plyrHnd->index;
        playr->plyrHnd->index++;      //Increment index for next card
        index++;                      //Increment index for next card choice in deck
        //Return face card and suit
        return deck[elem].face+deck[elem].suit;
}

//Function shuffles deck by using a random number to grab a face from
deck
//and swap it with deck[i]
```

```
void shuffle(Card deck[]){
	Card temp;		//Create a temp card to store face card
contents
	int r;		//Store random number
	for(int i=0;i<52;i++){
	r=rand()%52;		//Grab random number
	temp=deck[i];	//Store face card in temp card
	deck[i]=deck[r];	//Change face card in deck to random face card in
deck
	deck[r]=temp;		//Swap random face card with temp
	}
}

//Write to binary file
void wrtBin(fstream &bin, Player *plyr){
	//Write user structure to binary file
	bin.write(reinterpret_cast<char *>(plyr),sizeof(Player));
}

//Read from binary file
Player *readBin(fstream &bin){
	Player *record=new Player;
	bin.read(reinterpret_cast<char *>(record),sizeof(Player));
	return record;
}

void print(Player *plyr){
	cout<<"Game #: "<<plyr->games<<endl;
	cout<<"Status: "<<(plyr->status==true?"Win":"Loss")<<endl;
	cout<<"Bet Amount: "<<plyr->betAmt<<endl;
	cout<<"Chips Won: "<<plyr->chipsWon<<endl;
}

void endStats(Player *plyr){
	cout<<"\tTotal Games Played: "<<setw(5)<<plyr->games<<endl;
```

```
        cout<<"\tTotal Wins: "<<setw(13)<<plyr->wins<<endl;
        cout<<"\tTotal Losses: "<<setw(11)<<plyr->losses<<endl;
        cout<<"\tOverall chips won: "<<setw(6)<<plyr->totChips<<endl;
}
```