**Blackjack**

(Version 2)

Fredi Brizuela

CSC-5-40513

June 6, 2020

Table of Contents

Introduction 3

What is Blackjack? 3

**How to play 3**

**What are Splits and Doubling Down? 4**

My Approach to the Game 4

**Problems I faced 4**

**How I solved issues from my project 1 5**

**Why I revised my project 1 5**

How I approached this Project 5

**Pseudocode 5-7**

**Flowchart 7**

Concepts Utilized 8

**Tables listing all types of code used 8-10**

Proof of working product 10-12

References 12

Program 12

**Introduction**

Blackjack

The rules of blackjack are fairly simple and your only opponent in the game is the dealer. Each player is dealt two cards face up while the dealer only has one card face up. The goal is to have your two cards added total to be higher than the dealer’s two cards and under 21. If you go over 21 you “bust”, or lose, or if the dealers two cards added together are higher than yours you also lose. If your two-card total is equal to the dealers then it is a “stand-off” or a tie. Cards values are usually face value, 2 is 2 and 10 is 10, while all face cards, jack/queen/king, are also 10. The only exception to this rule is Aces, they can take on the value of 1 or 11. To get your two cards total you simply add your two cards together. If you have any combination of an Ace or any card that is 10 points then you have what is called blackjack, 21 in total. Getting blackjack means you get paid more if you win. With all of that being said if you’re not satisfied with your two-card total then you can take extra cards, called taking a hit, and for each hit you get you get one more card. The dealer does this as well but has a strict set of rules to follow on whether or not to take a hit. If a dealer’s hand is less than or equal to 16 or a “soft” 17, meaning they have an ace and a 6, they must take a hit. If it’s 17 or higher the dealer must stand. So now that we know the dealer rules there are a few options for users. As stated earlier you can take hits to increase your card total. You may also split your cards and double down. Splitting can be done when your first two cards are of equal value and can only be split from the original hand, split hands cannot be split, the bet has to be equal or greater than the original bet for each hand. For each time you split you will receive and additional card for that hand and then you play like regular blackjack. Users may also double down which consists of a user placing another bet of equal or lesser value when their first two cards total is equal to 9, 10, or 11 without aces. After doubling down, you will only get one additional card. Some of you may have realized that if the dealer gets a blackjack or 21 you pretty much always lose unless you yourself have blackjack. There is a way around this, and it’s called insurance. If the dealer is dealt an ace face up then the dealer will ask if you want to take out insurance, equal to half of your original bet, to insure your hand if the dealer has blackjack and only when he has blackjack and helps insure you don’t lose money if he does have blackjack, insurance pays 2 to 1 so your insurance bet will cover the loss of your hand if you bet half.

**My approach to the game**

Project Size: Approximately 550 Lines

The Number of variables: approximately 30 or more

Unlike my first project, this was one done with a great deal of time. In order to incorporate all of the things learned in the class, it was necessary to overhaul project 1. The reasoning was because of the prototype functions and the arrays would allow me to incorporate more elements such as names, passwords, and the ability to play the game using a currency that stays with player throughout the game.

**How I solved the issues**

Project 1 presented me with issues relating to Booleans and do and or while loops, hence why I choose to avoid them in my first project (as well as time constraint on my end). Regardless, looking at the previous lectures provided, I was able to understand and utilize both of those skills and incorporate them into prototype functions for efficiency. Other than that, it was important that I added more than my first project as it was barebones in nature. I added an array that allows for cards to be given a suit for more authenticity. As well as used vectors to provide a system that allows players who retry the ability to see their winnings and their initial bet and loss. I also avoided the use of Doubles and Global Constants as stated inn the instructions. Although it is a massive change to my original project, I feel it as though it is able to meet the standards given by incorporating many of the functions and variables taught in class with the exception of bubble sorts and or any type of sorts. Regardless, I was able to include an option to double the bet the player makes as well as the ability to split and to view aces as 1 or 11. An option was also given to view the rules stated above in the game to new players or to double check the rules.

**How I approached the project**

Link to GitHub repository

<https://github.com/FrediBrizuela/Project-2>

**Pseudo Code**

//Adding more in order to allow new code like a random generator or vectors

//types for the card suits Spades, Clubs, Diamonds, Hearts

//Structures for a score setter for players

//account structure. This will store all the player information

//The name on the account

//the username of the player

//password for the players account

//in order to add a reward or to keep progress in wins and loses

//number of wins a player has endured

//total of wins or loses a player has endured

//a storage for all of the cards being used in the game

//a ratio of how many games the player has played

//a storage for all of the cards being used in the game

//value of the cards

//Adding suits will make the game feel more similar to those in casinos

//for the dealer since one card is face down, something I did not include in my first project due to lack of time

//basis of each player in the game including the dealer

//player's account info in order to address them by their name

//the players hand

//how much the user is betting

//Prototype functions used throughout the project

//Dealer

//Hit or receive card

//Keep score of dealer and player

//In the case of an ace being present due to it having two values 1 or 11

//Print what suit the card is

//Prints Cards

//Double down on bet if player is certain of victory

//How much money the player has earned

//In the rare case that a player draws two cards of similar value, they can play both as if it were two players

//The Bet made by the player, in this case 5 dollars

//Loads account info if playing multiple times

//Saves player info for next games

//Allows player to create an account with password and username

//Allows the game to be played

//Clears the cards used to reset for a new game

//Dealers play if ace or split are involved

//Displays what happened if winner

//Menu for the game allowing for the player to view rules or make an account and play

//Displays a lengthy tutorial for Blackjack

//Program Execution Begins Here

//so, we can get a random number

//seeding the random number

//creates a player called user

//creates the dealer

//creates a vector of players

//for the Input

//sets up the dealer info

//puts the dealer in the vector

//To take in the input of the user

//displays the menu for the player

//will take the users input

//if the user wants to read the rules

//displays the rules

//if the user wants to play

//asks how many players, only one would suffice

//so, we can take in the players input

//takes in the input

//cycles through the players

//asks if they are a new or existing player

//takes in their input

//if they are new player…

//creates new account info

//saves the players info

//pushes info of player back into the vector

//if they have already played a game of blackjack

//pushes the info back in the vector

//if the user can't follow directions

//reiterates their only option in number of players

//resets menu in case of error

//ignores the buffer

//while they don't choose play, we will repeat this

//asking the user if they would wish to continue

//Verifies number of players which should be one

//Checks for players money

//checks to see if they player has any money in existing account

//if user is out of money, they have the chance to give themselves money

//takes in input

//how much money they would like to give themselves

//takes it in

//if not, we save their info and leave

//erases player from vector and game in order to create a new player

//if they have played at least once

//takes in input for the player quit or continue

//players want to quit

//erases player from vector

//Begins the game

//They finished one hand

//Ends Program

**Flowchart**

**A close up of a map

Description automatically generated**

**Concepts Utilized**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Frequency** | **Description** | **Location** |
| player.push | 4 | Moves info into the vector | throughout |
| player.size | 27 | Determine size of data for player | throughout |
| players[i] | 51 | Input data | throughout |
| players[i].info.money | 21 | Holds data for money for player | throughout |
| players.begin | 4 | Allows player to start the game with name and info | Line 129 |
| players.erase | 6 | Erases players info after game if needed | Line 132 |
| players.save | 3 | Saves the players info throughout the game | Line 365 and 367 |
| deal() | 34 | Allows for game to continue and creates cards | throughout |
| hitMe() | 32 | Lets player receive a card | throughout |
| score() | 12 | Keeps track of score | Line 561 |
| hasAce() | 3 | If ace is drawn, the program knows the value | Line 546 |
| pantSuit() | 10 | Allows for the suits to be identified | throughout |
| dbleDown() | 4 | Lets players increase their bet | Line 498 |
| payout() | 8 | Keeps track of payout for player | throughout |
| create() | 5 | Allows for program to name the player | Line 324 |
| bet() | 12 | Allows for a bet to be made | throughout |
| play() | 2 | Begins the game | Line 144 |

|  |  |  |  |
| --- | --- | --- | --- |
| clear() | 3 | Clears records of player that leaves the game | throughout |
| winner() | 5 | Keeps track of winner | throughout |
| acout create() | 12 | Creates account for user | throughout |
| disMenu() | 3 | Displays menu for players | Line 160 |
| rules() | 2 | Shows rules if necessary | Line 152 |
| setw() | 1 | Allows for width to be set | Line 192 |

**Proof of working product**

**Test Case 1**

**A close up of an object

Description automatically generated**

**Test Case 2**

**A close up of a logo

Description automatically generated**

**Test Case 3**

**A close up of a logo

Description automatically generated**

**Test Case 4**

**A screenshot of a social media post

Description automatically generated**

**Test Case 5**

**A screenshot of a cell phone

Description automatically generated**

**References**

1. Dr. Lehr’s Lectures & Lab

2. “Starting Out with C++: From Control Structures through Objects” Gaddis, Tony. 8th Edition. (Textbook)

3. www.cplusplus.com (for some of the variables I used)

4. [www.programiz.com](http://www.programiz.com) (on how to use some functions)

**Program**

/\*

\* File: main.cpp

\* Author: Fredi Brizuela

\* Created on June 6, 2020 9:38 PM

\* Purpose: Revised Blackjack, overhaul Project 1 to meet requirements, in order to meet requirements, most of my code will move and be revised

\*/

//System Libraries Here

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <cstdlib>

#include <ctime>

#include <iomanip>

//Adding more in order to allow new code like a random generator or vectors

using namespace std;

//types for the card suits Spades, Clubs, Diamonds, Hearts

enum card\_suit{

S, C, D, H

};

//Structures for a score setter for players

struct acout{ //account structure. This will store all the player information

string name; //The name on the account

string user; //the username of the player

string paswrd; //password for the players account

int money; //in order to add a reward or to keep progress in wins and loses

int wins; //number of wins a player has endured

int total; //total of wins or losses a player has endured

float winPerc; //a ratio of how many games the player has played

};

struct card{ //a storage for all of the cards being used in the game

int value; //value of the cards

card\_suit suit; //Adding suits will make the game feel more simular to those in casinos

bool up; //for the dealer since one card is face down, something I did not include in my first project due to lack of time

};

struct player{ //basis of each player in the game including the dealer

acout info; //player's account info in order to address them by their name

vector<card> hand; //the players hand

int bet; //how much the user is betting

};

//Prototype functions used throughout the project

card deal();//Dealer

card hitMe();//Hit or receive card

int score(vector<card>);//Keep score of dealer and player

bool hasAce(vector<card>);//In the case of an ace being present due to it having two values 1 or 11

char pntSuit(card);//Print what suit the card is

void pntCrds(vector<card>);//Prints Cards

void dblDown(player, player&);//Double down on bet if player is certain of victory

void payout(player, player&);//How much money the player has earned

void split(player&, player&);//In the rare case that a player draws two cards of simular value, they can play both as if it were two players

void bet(player&);//The Bet made by the player, in this case 5 dollars

acout load();//Loads account info if playing multiple times

void save(player);//Saves player info for next games

acout create();//Allows player to create an account with password and username

void play(vector<player>&);//Allows the game to be played

void clear(vector<card>&);//Clears the cards used to reset for a new game

void deal\_ply(player&);//Dealers play if ace or split are involved

int winner(player, player&);//Displays what happened if winner is chosen

void disMenu();//Menu for the game allowing for the player to view rules or make an account and play

void rules();//Displays a length tutorial for Blackjack

//Program Execution Begins Here

int main(){

unsigned int time\_seed = time(0); //so we can get a random number

srand(time\_seed); //seeding the random number

player user; //creates a player called user

player dealer; //creates the dealer

vector<player> players; //creates a vector of players

char input; //for the Input

dealer.info.user = "Dealer"; //sets up the dealer info

players.push\_back(dealer); //puts the dealer in the vector

int menu; //To take in the input of the user

do{

disMenu(); //displays the menu for the player

cin>>menu; //will take the users input

if(menu == 1){ //if the user wants to read the rules

rules(); //displays the rules

}

else if(menu == 2){ //if the user wants to play

cout << "How many players(1)" << endl; //asks how many players, only one would suffice

int numPlyr; //so we can take in the player

cin >> numPlyr; //takes in the input

for(int i = 0;i< numPlyr;i++){ //cycles through the players

cout << "Are you a new player(Y)?" << endl; //asks if they are a new or existing player

cin >> input; //takes in their input

if(input == 'Y' || input == 'y'){ //if they are new player...

user.info = create(); //creates new account info

save(user); //saves the players info

players.push\_back(user); //pushes info of player back into the vector

}

else if(input == 'N' || input == 'n'){ //if they have already played a game of blackjack

players.push\_back(user); //pushes the info back in the vector

}

}

}

else{ //if the user can't follow directions

cout << "Please enter (1)" << endl; //reiterates their only option in number of players

menu = 0; //resets menu in case of error

cin.ignore(); //ignores the buffer

}

}while(menu != 2); //while they don't choose play we will repeat this until we finish the never ending story

bool cond = false; //asking the user if they would wish to continue

int size = players.size();//Verifies number of players which should be one

do{

for(int i =1;i<players.size();i++){//Checks for players money

if(players[i].info.money < 5){ //checks to see if they player has any money in existing account

cout << "You have no money! Would you like to add some money, yes(Y) or (N)?" << endl; //if user is out of money, they have the chance to give them self money.

cin >> input; //takes in input

if(input == 'Y' || input == 'y'){ //if they say yes

cout << "How much would you like to add?" << endl; //how much money they would like to give themselves

cin >> players[i].info.money; //takes it in

}

else{ //if not we save their info and leave

save(players[i]); //saves info

players.erase(players.begin() + (i)); //erases player from vector and game in order to create a new player

i--;

}

}

if(cond && players.size() > 1){//if they have played at least once

cout << players[i].info.user << " Would you like to continue playing Blackjack(Y) or not(N)" << endl;

cin >> input; //takes in input for the player quit or continue

if(input == 'n' || input == 'N'){ //player want to quit

save(players[i]); //saves info for next game

players.erase(players.begin() + (i)); //erases player from vector

i--;

}

}

}

play(players); //Begins the game

cond = true; //They finished one hand

}while(players.size() > 1);

//!(input == 'n' || input == 'N')

return 0;//Ends Program

}

//Displaying the rules

void rules(){

cout<< "The rules of blackjack are fairly simple and your only opponent in the game is the dealer. Each player is dealt a two cards face up while the dealer only has one card face up. The goal is to have your two cards added total to be higher than the dealer’s two cards and under 21. If you go over 21 you “bust”, or lose, or if the dealers two cards added together are higher than yours you also lose."<<endl;

cout<< "If your two card total is equal to the dealers then it is a “stand-off” or a tie. Cards values are usually face value, 2 is 2 and 10 is 10, while all face cards, jack/queen/king, are also 10. The only exception to this rule are Aces, they can take on the value of 1 or 11. To get your two cards total you simply add your two cards together. If you have any combination of an Ace or any card that is 10 points then you have what is called blackjack, 21 in total. Getting blackjack means you get paid more if you win."<<endl;

cout<< "With all of that being said if you’re not satisfied with your two card total then you can take extra cards, called taking a hit, and for each hit you get you get one more card. The dealer does this as well but has a strict set of rules to follow on whether or not to take a hit. If a dealer’s hand is less than or equal to 16 or a “soft” 17, meaning they have an ace and a 6, they must take a hit. If it’s 17 or higher the dealer must stand. So now that we know the dealer rules there are a few options for users."<<endl;

cout<< "As stated earlier you can take hits to increase your card total. You may also split your cards and double down. Splitting can be done when your first two cards are of equal value and can only be split from the original hand, split hands cannot be split, the bet has to be equal or greater than the original bet for each hand. For each time you split you will receive and additional card for that hand and then you play like regular blackjack. Users may also double down which consists of a user placing another bet of equal or lesser value when their first two cards total is equal to 9, 10, or 11 without aces."<<endl;

cout<< "After doubling down you will only get one additional card. Some of you may have realized that if the dealer gets a blackjack or 21 you pretty much always lose, unless you yourself have blackjack. There is a way around this and it’s called insurance. If the dealer is dealt an ace face up then the dealer will ask if you want to take out insurance, equal to half of your original bet, to insure your hand if the dealer has blackjack and only when he has blackjack and helps insure you don’t lose money if he does have blackjack, insurance pays 2 to 1 so your insurance bet will cover the loss of your hand if you bet half." << endl; //displays the rules of blackjack

}

//Displays Menu

void disMenu(){

cout << "Welcome to Blackjack!" << endl; //Displays welcome

cout << "Please choose one of the two options below." << endl; //Asks user to select 1 or 2

cout << "1) Rules of Blackjack" << endl; //Rules

cout << "2) Play Blackjack" << endl; //Continue with the menu

}

//Lets the dealer play as one does in a casino

void deal\_ply(player &dealer){

if((score(dealer.hand) < 17) || (score(dealer.hand) == 17 && hasAce(dealer.hand))){ //dealer hits at less than 17

dealer.hand.push\_back(hitMe()); //dealer gets a card because of house rule

}

}

// How the game will be played, the code will take care of all the play portions of the game. Taking hits, splitting, etc. Takes in the vector of players by reference so we can make changes directly to the players for future games

void play(vector<player> &players){

char input; //for our input

for(int i =1;i<players.size();i++){ //cycles through the player and dealer

bet(players[i]); //players bet here

cout << "Money: " << players[i].info.money << setw(10) << "Bet: " << players[1].bet << endl; //display their money and bet amount

}

//Gives the players (dealer included) one card at a time starting with the dealer

for(int i = 0;i< (players.size()\*2);i++){

players[(i%players.size())].hand.push\_back(deal());

if((i%players.size()) == 0 && (i%2) == 0){ //the dealers first card

players[(i%players.size())].hand[(i%2)].up = false; //is set to false since it's face down

}

}

//Displays the score for the player

for(int i=1;i<players.size();i++){

cout << players[i].info.user << " has: " << score(players[i].hand) << setw(10) << endl;

}

//Function for the card being used by the player

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

cout << players[i].info.user << " Cards:" << endl;

pntCrds(players[i].hand);

}

//The below function actually consists of the playing Blackjack

bool cont = true; //to loop the game mechanics

for(int i = 1;i<players.size();i++){ //cycles through the player since each player plays on their own card

do{

if(players[0].hand[1].value == 1 && cont){ //if the dealer has an ace and cont is true, only happens the first time if the dealer doesn't have blackjack

if(score(players[0].hand) == 21){ //checks to see if the dealer has blackjack

players[0].hand[0].up = true; //if they do, set the first card to face up

pntCrds(players[0].hand); //prints the dealers cards

//The below function cycles through the players and pays them out since they lost

for(int i =1;i<players.size();i++){

payout(players[0],players[i]);

}

input = 'S'; //sets input to stay since they just lost

}

cont = false; //if the dealer didn't have blackjack this is now false

}

if(players[0].hand[1].value >= 10 && cont){ //if the dealer has a 10 or face card showing they have blackjack that's game

if(score(players[0].hand) == 21){ //if they have blackjack

players[0].hand[0].up = true; //puts the dealers first card face up

pntCrds(players[0].hand); //prints the dealers card

//The below function pays out the player since they just lost

for(int i =1;i<players.size();i++){

payout(players[0],players[i]);

}

input = 'S'; //input is now S since the players lost

}

cont = false; //if the dealer doesn't have 21 we don't care about this anymore

}

//As long as the players score is less than 21

if(score(players[0].hand) <= 21){

// If they have a pair of 5's they can split, double down, hit, or stay

if(((players[i].hand[0].value >= 10 && players[i].hand[1].value >= 10) || players[i].hand[0].value == players[i].hand[1].value) && players[i].hand.size() == 2 && score(players[i].hand) == 10){

cout << players[i].info.user << " score: " << score(players[i].hand) << endl; //shows them their score

cout << "Would you like to Double Down(D), split(L),take a hit(H), or stay(S), default is to take a stay?" << endl; //ask them

}

else if(((players[i].hand[0].value >= 10 && players[i].hand[1].value >= 10) || (players[i].hand[0].value == players[i].hand[1].value)) && players[i].hand.size() == 2){ //if they can split their cards

cout << players[i].info.user << " score: " << score(players[i].hand) << endl; //shows them their score

cout << "Would you like to split(L) your cards, take a hit(H), or stay(S), default is to take a stay?" << endl; //ask them

}

else if(players[i].hand.size() == 2 && score(players[i].hand) >= 9 && score(players[i].hand) <= 11 && !(hasAce(players[i].hand))){ //can they double down

cout << players[i].info.user << " score: " << score(players[i].hand) << endl; //shows them their score

cout << "Would you like to Double Down(D), take a hit(H), or stay(S), default is to take a stay?" << endl; //asks them

}

else{ //they can't do anything

cout << players[i].info.user << " score: " << score(players[i].hand) << endl; //shows them their score

cout << "Hit(H) or Stay(S), default is to take a stay?"; //asks them what they want to do

}

cin >> input; //takes in the input

switch(input){ //what did they choose?

case 'L': //they wanted to split

split(players[0], players[i]); //we split them

pntCrds(players[i].hand); //reprint their cards in case they forgot

break;

case 'D':

dblDown(players[0], players[i]); //they double down

input = 'S'; //sets input to S since now they are done

break;

case 'H':

players[i].hand.push\_back(hitMe()); //we give them one more card for their hit

pntCrds(players[i].hand); //reprint their cards

cout << players[i].info.user << " score is now " << score(players[i].hand) << endl; //reprint their score

break;

default: //this is here for people can't follow directions

input = 's'; //input is S

}

if(score(players[i].hand) > 21){ //if they bust they are done

input = 'S'; //so we can quit

}

}

}while(!(input == 'S' || input == 's')); //we continue doing this until they are want to stay

}

deal\_ply(players[0]); //now the dealer plays

players[0].hand[0].up = true; //now the everybody can see the first card

//The below method shows everybody's score and cards including dealers

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

cout << players[i].info.user << " score: " << score(players[i].hand) << " Cards: ";

pntCrds(players[i].hand);

}

//The below method pays everybody out

for(int i =1;i<players.size();i++){

if(score(players[i].hand) > 21){ //if the player busted we tell them

cout << "You busted! ";

}

int win = winner(players[0], players[i]); //we figure out who wins

if(win == 1){

players[i].info.wins += 1; //if the player wins it adds one to their win record

}

payout(players[0],players[i]); //payout everybody

clear(players[i].hand); //we clear out their hands

players[i].info.total+=1; //adds one to the total played

}

clear(players[0].hand); //clear out the dealers hand

}

//The below method clears out a players hand for the next game

void clear(vector<card> &hand){

hand.clear();

}

//Creates new accounts for new players

acout create(){

acout user; //creates a new account

cout << "What is the username you'd like to use?" << endl;

fstream input; //so we can take in the input

string filename; //what file we will open

do{

cin >> user.user; //takes in the username

filename = user.user + ".dat"; //adds the extension to the file

input.open(filename); //opens the file

if(!input.fail()){ //if it didn't fail the name is taken

cout << "Error." << endl; //we tell the user

}

}while(!input.fail()); //repeat until it doesn't fail

cout << "Please enter a password." << endl; //asks for password

cin.ignore(); // so cin will work

getline(cin, user.paswrd); //takes in the line b/c it may be more than one word

cin.ignore(); //so the next line will wokr

cout << "Please enter your name." << endl; //asks for the users name

getline(cin, user.name); //takes it in

cout << "Please enter how much money you'd like to deposit" << endl; //asks for money

cin >> user.money; //takes it in

user.total = 0; //sets these to 0

user.wins = 0; //sets these to 0

user.winPerc = 0; //set these to 0

input.close(); //closes the input stream

return user; //returns the account

}

//For saving account info for when you're done

void save(player user){

ofstream output; //output stream

string filename = user.info.user + ".dat"; //so we know what file to save to

output.open(filename, ios::out); //we want to erase all previous data

if(!output){//if we can't open output

cerr << "Output file will not open and data could not be saved. " << endl;

}

float percent = 0; //for the win percentage

if(user.info.total == 0){

percent = 0; //so we don't divide by zero

}

else{

percent = ((float)user.info.wins/user.info.total)\*100; //win percentage

}

output << user.info.user << endl << user.info.paswrd << endl << user.info.name << endl << user.info.money << endl << user.info.total << endl << user.info.wins << endl << setprecision (3) << percent; //writes the data to the file

output.close(); //closes the output stream

}

//For loading account info

//Asks the user what they would like to bet and returns it

void bet(player &user){

int bet;

do{

cout << user.info.user << endl; //so we know what player is betting

cout << "How much would you like to bet? (Must be greater than 5 and less than " << user.info.money << ")" << endl; //we tell them what they can bet

cin >> bet; //takes in their bet

}while(!(bet >= 5 && bet <= user.info.money)); //repeat until they get it right

user.info.money -= bet; //subtract the bet from their money stock pile

user.bet = bet; //set their bet

}

//If the user chooses to split their cards the split hand is completely played here

void split(player &dealer, player &user){

player split;

vector<player> players; //creates a new player for the split

players.push\_back(dealer);

split.bet = user.bet; // takes the extra bet

user.info.money-= user.bet; //takes out the bet from the user money pile

split.hand.push\_back(user.hand[0]); //takes the first card from the user and gives it to the split player

split.hand.push\_back(deal()); //gives the split person a new card

user.hand[0] = deal(); //gives the user a new card

split.info.user = user.info.user;

players.push\_back(split); //pushes the split player onto the vector

pntCrds(players[1].hand); //prints out the new cards

char input; //what the input is

do{

for(int i =1;i<players.size();i++){

if(score(players[i].hand) > 21){ //if they bust

input = 'S'; //they are done

}

else{

cout << "Hit(H) or Stay(S):"; //otherwise we ask them if they want to take a hit

cin >> input; //take in their input

}

/\*\*

If they take a hit the below code gives them a new card, prints out their new cards and their new score

\*/

if(input == 'H' || input == 'h'){

players[i].hand.push\_back(hitMe());

pntCrds(players[i].hand);

cout << players[i].info.user << " score is now " << score(players[i].hand) << endl;

}

}

}while(!(input == 'S' || input == 's')); //we repeat this until they stop taking hits or bust

deal\_ply(players[0]); //dealer plays now since we need to take care of this now rather than later

//prints the split players cards and score again

for(int i =1;i<players.size();i++){

cout << players[i].info.user << " score: " << score(players[i].hand) << " Cards:" << endl;

pntCrds(players[i].hand);

}

//payouts the split player

for(int i =1;i<players.size();i++){

if(score(players[i].hand) > 21){

cout << "You busted!"; //tells them they busted

}

payout(players[0],players[i]); //plays out the player

}

}

//Typical pay out rules. If the player wins he gets 2 to 1 odds.

void payout(player dealer, player &user){

if(winner(dealer, user) == 1){ //if the player won

if(score(user.hand) == 21 && hasAce(user.hand) && user.hand.size() == 2){ //if the player has blackjack it's a 3:2 payout

user.info.money += ((user.bet\*3)/2); //I think this is 3:2 odds

user.bet = 0; //clears out the bet

cout << user.info.user << " won!" << endl;

}

else{

user.info.money+= (user.bet\*2); //adds the bet to the players stash of cash

user.bet = 0; //clears out the bet

cout << user.info.user << " won!" << endl;

}

}

else if (winner(dealer, user) == 0){ //they tied

user.info.money+= user.bet; //players money goes back to his pile

user.bet = 0; //clears out the bet

cout << user.info.user << " tied!" << endl;

}

else{ //the dealer won.

user.bet = 0; //player didn't win so all we need to do is clear out.

cout <<user.info.user << " lost!" << endl;

}

}

//Figures out if the dealer or the user won, returns 1 if the player won, returns -1 if the dealer won, returns 0 if they tie

int winner(player dealer, player &user){

if(score(dealer.hand) == score(user.hand)){ //they had a stand off

return 0;

}

else if(((score(dealer.hand) < score(user.hand)) && (score(user.hand) <= 21)) || (score(dealer.hand) > 21 && score(user.hand) <= 21)){ //user won

return 1;

}

else{ //dealer typically wins

return -1;

}

}

//In the case the player wants to double down

void dblDown(player dealer, player &user){

int bet; //so we can store the new bet

do{

cout << "How much would you like to bet? (Must be greater than 5 and less than "<< user.info.money <<")" << endl; //asks how much they want to bet

cin >> bet; //takes it in

}while(!(bet > 5 && bet <= user.bet && bet <= user.info.money)); //we repeat this until they get it right

user.bet+=bet; //add the new bet to the original

user.info.money-=bet; //takes out their double down bet from the money

user.hand.push\_back(deal()); //gives the user one more card

payout(dealer, user); //pays out the player based on if he/she won

}

/\*\*

Returns the Character version of the suit

Fairly self-explanatory...takes in a number between 0-4 and returns the suit based off that number

\*/

char pntSuit(card new\_card){

switch(new\_card.suit){

case 0:

return 'S';

case 1:

return 'H';

case 2:

return 'D';

case 3:

return 'C';

}

}

/\*\*

Prints the cards to the screen

\*/

void pntCrds(vector<card> hand){

const string CARD\_VALUES[14] = {"A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "X"}; //makes it easier to print

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

if(hand[i].up){ //if the hand is face up we print this

cout << CARD\_VALUES[(hand[i].value-1)] << pntSuit(hand[i]) << " ";

}

else{ //if it's face down we print XX

cout << CARD\_VALUES[13] << CARD\_VALUES[13] << " ";

}

}

cout << endl;

}

/\*\*

Lets us know if the hand has an ace

\*/

bool hasAce(vector<card> hand){

bool has\_ace = false; //For now we say there is no ace in the hand

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

if(hand[i].value == 1){ //we have an ace

has\_ace = true; //so we set this to true

}

}

return has\_ace;

}

/\*\*

Gets the score for the user

Aces are seen as 1 and then later check to see if the hand contains an Ace

\*/

int score(vector<card> hand){

int total = 0; //setting up the total value

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

if(hand[i].value >= 10){ //if it's 10, J, Q, or K

total+=10; //adds 10 to the toal

}

else{

total += hand[i].value; //adds the value to the total

}

}

if(hasAce(hand) && total <= 11){ //if the hand has an ace and we won't bust

total+=10; //add 10

}

return total; //return the total

}

//gets a new card for the player

card hitMe(){

return deal(); //add another card to the players hand

}

//takes in nothing and returns a card, makes a new card and assigns it a random value between 1-13

card deal(){

card new\_card; //card we will be returning

new\_card.value = 1 + rand() % 13; //makes sure the random number is between 1 and 13 for the value

int suit = rand() % 4;

switch(suit){ //makes sure the random number is between 1 and 4 for the suit

case 0:

new\_card.suit = S;

break;

case 1:

new\_card.suit = H;

break;

case 2:

new\_card.suit = D;

break;

case 3:

new\_card.suit = C;

break;

}

new\_card.up = true; //we'll change it later if it's the dealers face down card

return new\_card; //returning the card

}