CS 210 - Fundamentals of Programming I
Spring 2017 - Programming Project 7
40 points

Out: April 16, 2019
Due: April 30, 2019 (Tuesday, last day of class)

Note that this project is worth more toward the final grade than previous projects.

Problem Statement
Many people like to visit casinos for entertainment. One of the common games available is Blackjack. In order to increase our chances of winning, we would like a program to help us practice playing the game.

Blackjack Game
You are encouraged to look at the Wikipedia entry for a fuller description of the game. Here are the essentials:

- Blackjack is played with cards having four suits, Spades, Hearts, Diamonds, and Clubs, each with 13 possible values , Ace, Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, and King.
- A blackjack hand starts with two cards each for you and the dealer. The game points of the cards are 10 for face cards (Jack, Queen, King), 1 or 11 for Aces, and the number of all the rest of the cards (e.g., 2 for Two).
- The goal is to try to get a total as close to 21 as possible without going over. Note: since Aces can be either 1 or 11, you pick the largest value which doesn't exceed 21, if possible.
- The only options you have are to either *hit*, which means to add another card to your hand, or to *stand* which ends your turn.
- You are playing against the dealer. At the beginning of the game, you only get to see one of the dealer's cards. After you are done playing, the dealer reveals the second card and plays his turn. The dealer must follow this rule: While the dealer's total is less than 17, he must hit, otherwise, he must stand.
- If you go over 21, called being *busted*, you lose immediately (**before the dealer draws more cards**).
- Otherwise, if the dealer goes over 21, you win.
- Otherwise, the person with the larger total wins. If the totals are the same it's a tie.

Blackjack is a gambling game, so before each hand you make a bet. If you win, you get the double the amount of your bet back. If you lose, you get nothing. If there's a tie, you get your original bet back, but with no extra.

Program Description
The sample run shows a typical interaction with this program. Since there is a random element to this game, the submission system actually will be checking your implementation of the card playing library. The instructor will check your actual game by hand.

In our program, the user will start with $1,000. The game will end either when the player runs out of money, or signifies that they are done playing by placing a negative bet. Also, typical casino blackjack games use multiple decks of cards (called a *shoe*) to make it more difficult to count cards. At the beginning of the program, the user is asked how many decks they wish to play with. This program will allow up to 10 decks to be used.

Program Design
This program is required to use several different structs to represent the different data types in this program, along with functions that perform operations on them. These functions are what the submission system will test.

Cards

Cards are represented by a structure named `card_t`. It must have two fields representing the suit (a character) and the value (an integer) of the card.

Card Functions

`card_create` This function receives a suit character and an a card value, and returns a `card_t` struct. It should accept upper or lowercase for the suit and convert lowercase to uppercase (using `toupper()`). If the suit character is not one of '`S`' (for Spades), '`H`' (for Hearts), '`D`' (for Diamonds), or '`C`' (for Clubs), the function should set the suit field to '`C`'. The card value must be 1 (for Ace), 2 (for Two), etc., up to 13 (for King). If the card value is not 1-13, the function should set the card value field to 2. (Thus the default card value is the Two of Clubs.)

`card_get_suit` This function receives a single card and returns a character representing the suit of the card. It should return '`S`' for Spades, '`H`' for Hearts, '`D`' for Diamonds, or '`C`' for Clubs.

`card_get_value` This function receives a single card and returns an integer between 1 and 13 representing the value of Ace through King, respectively, of the card.

`card_blackjack_points` This function receives a single card and returns the Blackjack game points for this card . Since Aces can be either 1 or 11, this function always will return 1. The logic for computing the Blackjack game points of an entire hand will take the 11 case into account.

`card_print` This function receives a single card and prints out a long description of it on a single line. (e.g. "Ace of Hearts", "7 of Diamonds") with no trailing newline.

Shoe

The shoe is a "deck" of all the cards. It will be represented by a structure named `shoe_t`. It will have 3 fields. The first field is a ***dynamic*** array of `card_t`'s. The second field is an integer that represents the total number of 52-card decks that make up the full shoe. The third field is an integer to keep track of the next card that will be dealt out of the shoe. The easiest way to do this is to not actually remove the card from the array, but move this index to the next card that will be dealt.

Shoe Functions

`shoe_create` This function receives an integer that is the number of decks in the shoe and returns a shoe that is completely initialized with all the cards and shuffled. It must allocated a dynamic array exactly large enough to hold all of the cards.

`shoe_draw_card` This function receives and passes back a shoe, draws the next card from it and returns it. The shoe must be passed back, since the act of drawing a card must change the next card field. Note, if there are no cards left to draw, reshuffle the deck, and draw the first card of the shoe.

`shoe_cards_left` This function receives a shoe and returns the number of cards that are left to be dealt from it before it must be reshuffled.

`shoe_reshuffle` This function receives and passes back a shoe and randomizes the cards within it and resets the next card field to the first card. It should print out "`SHUFFLING`", so the card counters will know that everything is reset. To shuffle an array, you have to swap each element with a random element. You can use the following ***pseudocode*** algorithm that uses the built-in random number generator `rand()` function to shuffle an array `arr` with `n` elements:

```
for(i=0; i<n; i++) {
   random_position = i + rand()%(n-i);
   swap arr[i] with arr[random_position]
}
```

The random number generator will always generate the same sequence unless you "seed" it. The code to do this is:

```
srand(time(0));
```

This seeding should be done at the beginning of the **main program**. You will need to include the `<time.h>` library for this to work.

`shoe_free` This function receives and passes back a shoe and frees the dynamic array within it. This will only need to be called once at the end of your program.

Hand

The hand is all cards that a player (or dealer) currently has. It will represented by a structure named `hand_t` that has two components. The first component is an array of `card_t` cards that are the actual cards. Since there can never be more than 21 cards in a legal blackjack hand, we don't need to use dynamic arrays, we can simply set the maximum size to 21. The second component is an integer representing the number of cards actually in the array. By combining these two fields into a single struct, we won't have to constantly pass two arguments around.

Hand Functions

`hand_create_empty` This function returns an empty hand (simply has the number of cards set to 0). This is useful when initializing variables.

`hand_print` This function receives a hand and prints each card in the hand on a separate line.

`hand_print_first_card` This function receives a hand and prints the first card of the hand on a line including a new line.

`hand_add_card` This function has two parameters. The first parameter is the hand to add a card to that is received and passed back. It must be by passed back because we are actually changing its internal values. The second parameter is the `card_t` card that will be added to the hand.

`hand_blackjack_points` This function receives a hand and returns an integer that is the Blackjack game point value of the hand. Since Aces can be a bit tricky, here's a hint: start off counting them as 1 point, then if there's an Ace and the total is less than or equal to 11, add 10 points. Note that only one Ace will ever be counted as 11 points.

The Main Program

Given all of the above functions, the main function is somewhat straightforward. Here is a ***rough*** outline of the game:

1. Print the greeting and initialize the player's stake to $1,000.
2. Ask the user for the number of decks that will be in the shoe for this game (make sure that only legal numbers are allowed) and create a shoe of that size.
3. While the player still has money and hasn't quit by entering a negative bet:
   (a) Print the remaining stake and the number of cards left in the shoe.
   (b) Ask the player for the bet (and validate it, i.e. it is less than or equal to the player's current stake).
   (c) Draw the dealer's initial hand and player's initial hand.
   (d) Display only the first card of the dealer's hand and the entire player's hand
   (e) While the player's score is less than 21 and they haven't stood:
      i. Ask the user to stand or hit (and validate the input).
   (f) Determine the winner (dealer may or may not need to draw cards first) and update the player's stake.
4. Print the final results of the games.

Coding Notes

- **The names of the struct types and functions and order of parameters just match exactly** as given above, otherwise the submission system will not be able to compile the test program.
- The struct definitions and the function prototypes must be put in a file named (exactly) **`cards.h`** and the function definitions must be put in a file named (exactly) **`cards.c`**. Both **`cards.c`** and the main program file **`main.c`** will need to include the **`cards.h`** file.
- Recall that reference parameters are pointers to the actual argument. When using a pointer to a struct, to access a field use the "arrow" operator (**`->`**) instead of the "dot" operator.
- For good coding practice, only the functions of a structure type should be used to access and manipulate data of that structure type. E.g., the card functions can (and must) access the suit and value components of the **`card_t`** struct, but everywhere else (hand and shoe functions, main program) should use the card accessor functions. The reason for this is that a programmer using the structure type should not need to know what the component names are to use it.

REMINDER: Your program must compile for it to be graded. Submissions that do not compile will be returned for resubmission and assessed a late penalty. Submissions that do not substantially work also will be returned for resubmission and assessed a late penalty.

Follow the program documentation guidelines in the *C Programming Style Guideline* handout. As stated in the syllabus, part of the grade on a programming assignment depends on how well you adhere to the guidelines. The grader will look at your code and grade it according to the guidelines.

What to Submit

Electronically submit a zipfile containing **`main.c`**, **`cards.h`**, and **`cards.c`** as explained in the handout *Submission Instructions for CS 210*. The submission system will start accepting assignments no earlier than the evening of Monday, April 22.

Sample Runs (user input shown in **bold**)

```
Welcome to Blackjack!

How many decks do you wish to play with (1-10)? 1

SHUFFLING!
Your stake: $1000
Cards left in the shoe: 52

Enter your bet (negative to quit): 1000
The Dealer is showing 3 of Hearts

Your cards:
6 of Spades
3 of Clubs
Score = 9
Do you want to (H)it or (S)tand? h
Your cards:
6 of Spades
3 of Clubs
7 of Hearts
Score = 16
```

```
Do you want to (H)it or (S)tand? s
Dealer cards:
3 of Hearts
8 of Diamonds
9 of Diamonds
Dealer wins with a score of 20. You Lose!
You've lost your entire stake!


Welcome to Blackjack!

How many decks do you wish to play with (1-10)? 2

SHUFFLING!
Your stake: $1000
Cards left in the shoe: 104

Enter your bet (negative to quit): 200
The Dealer is showing 5 of Hearts

Your cards:
9 of Hearts
2 of Clubs
Score = 11
Do you want to (H)it or (S)tand? h
Your cards:
9 of Hearts
2 of Clubs
8 of Hearts
Score = 19
Do you want to (H)it or (S)tand? s
Dealer cards:
5 of Hearts
9 of Clubs
Ace of Clubs
9 of Spades
Dealer Busts. You Win $400!
Your stake: $1200
Cards left in the shoe: 97

Enter your bet (negative to quit): 800
The Dealer is showing Jack of Spades

Your cards:
2 of Clubs
King of Spades
Score = 12
Do you want to (H)it or (S)tand? h
Your cards:
2 of Clubs
King of Spades
2 of Diamonds
```

```
Score = 14
Do you want to (H)it or (S)tand? h
Your cards:
2 of Clubs
King of Spades
2 of Diamonds
9 of Diamonds
Score = 23
You Busted. You Lose!
Your stake: $400
Cards left in the shoe: 91

Enter your bet (negative to quit): 200
The Dealer is showing 4 of Diamonds

Your cards:
Queen of Spades
10 of Spades
Score = 20
Do you want to (H)it or (S)tand? s
Dealer cards:
4 of Diamonds
9 of Diamonds
7 of Hearts
It's a tie. You get back $200.
Your stake: $400
Cards left in the shoe: 86

Enter your bet (negative to quit): 400
The Dealer is showing 10 of Diamonds

Your cards:
Queen of Spades
Ace of Spades
Score = 21
Do you want to (H)it or (S)tand? s
Dealer cards:
10 of Diamonds
9 of Hearts
You win with a score of 21!  You win $800
Your stake: $800
Cards left in the shoe: 82

Enter your bet (negative to quit): -1
You ended the night with $800
```