CSCI.4430/6430 Programming Languages Fall 2014
Programming Assignment #2

This assignment is to be done either individually or in pairs. Do not show your code to any other group and do not look at any other group's code. Do not put your code in a public directory or otherwise make it public. However, you may get help from the TAs or the instructor. You are encouraged to use the LMS Discussions page to ask questions so that other students can also answer/see the answers.

Blackjack Game Simulation using Functional Programming

1. Goal

The goal of this assignment is to simulate a version of the Blackjack game, one of the most popular games played in the casinos. Specifically, you will program the dealer and the players at the table.

2. Blackjack Game

The game begins with the dealer shuffling a single standard deck of 52 playing cards. The game then proceeds in rounds. In every round, each player places a bet amount on the table before any cards are dealt. The dealer then deals one open card to every player on the table and one card for herself. She then deals the second open card to all the players and a closed card for herself. Once the initial cards are dealt, each player plays the round with the dealer separately.

2.1 Player vs Dealer

2.1.1 Value of a Hand

The value of cards 2,3,...,10 is equal to their rank. The value of a King, Queen, or Jack is always 10. An Ace can be counted as either 1 or 11. The value of a set of cards (hand) is equal to the sum of the value of the individual cards. Note that the suit of a card has no impact on its value.

2.1.2 Player's Options

In a round, a player chooses one of the following 3 options:

1. **Stand**: End the round.
2. **Hit**: Get another card from the dealer.
3. **Double down**: Hit then stand and double your bet.

Until he either stands or reaches 21 points or higher.

2.1.3 Dealer's Play

Once all the players finish their play, the dealer reveals her closed card. She continues to draw cards until the first time the value of her hand can be greater than or equal to 17.

2.2 Who Wins

A player/dealer is said to be busted if the value of his/her hand is > 21. The winning amounts in a given round are settled as follows :

1. Busted players always lose their bet amount.
2. If the dealer busts then all the non-busted players win the round.
3. Otherwise, a player wins if the value of his hand is greater than that of the dealer.
4. On a tie no money is exchanged.
5. The winning amount for a player is equal to the bet amount if the hand value is not 21. He gets a bonus equal to the bet amount if it is exactly 21 (a Blackjack.)

2.3 End of the round

After all the winning amounts are settled, the dealer collects all the cards dealt in that round and piles them in a different stack. Cards should be collected one player at a time in the order they were dealt, followed by the dealer's cards. In future rounds, whenever the dealer runs out of cards to deal, she shuffles the cards from the piled up stack and places them in the play stack.

3. Assignment

Your goal is to simulate the Blackjack dealer and multiple players in the functional programming subset of the Oz programming language. This means that you should not use objects or explicit state, only functions. Furthermore, you should not use `for` and `while` loops for control flow, but instead use recursion.

Each player starts with a fixed initial amount and bets a certain amount each round. All the players play with the exact same strategy. The table shows the player choices depending on the current value of their hand.

Current hand value (v)	Player Move
v≤8	Hit
9≤v≤11	Double down if the player has enough money, otherwise hit.
12≤v≤16	Hit.
17≤v	Stand.

Note that the value of an ace card should be treated as 11 if that would result in the total hand value being 21 or less and 1 otherwise (if 11 would bring the value of the hand over 21.)

3.1 Part 1

1. Create a representation of the deck and a function to initialize it.
   - You may use any representation you choose however you must describe your representation in your README file.
   - Whatever representation you choose the deck of cards must start in the following order:
     - First all clubs, then all diamonds, followed by all hearts, ending in all spades.
     - The cards for a given suit must be ordered: Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King.
2. Write a function which shuffles the deck.
   - The following code is a pseudo-random number generator (see CTM 6.8.3 on Pg. 474 for its usage.) This is the only piece of non-functional code that should be in your program. It should not be changed from the given form and should be used for your shuffling function to achieve deterministic results. It should be initialized with the seed provided as input to your program.

     ```
     local A=333667 B=213453321 M=1000000000 in
       proc {NewRand ?Rand ?Init ?Max}
         X = {NewCell 0} in
         fun {Rand} X := (A*@X+B) mod M end
         proc {Init Seed} X := Seed end
         Max = M
       end
     end
     ```
   - The shuffle function must move from the position of the first card to the position of the last card in the deck swapping each card with a random card anywhere in the deck. This function should only run once to shuffle a deck. Some cards may be swapped more then once due to previously being swapped into the position currently selected for swapping.
3. Write a function to compute the value of a hand.

3.2 Part 2

Write a main function that simulates the Blackjack game play between the dealer and the players. It should be possible to call your function in the following way: `{Browse {BlackJack S N M B R}}`

Parameter Name	Parameter Meaning
S	Seed for the pseudorandom number generator
N	Number of players at the table
M	Initial amount of money for each player
B	Amount that each player bets per round
R	Maximum number of game rounds to simulate

3.2.1 Assumptions

Make the following assumptions in your program.

1. The players are kicked from the game when they have less than B money to bet.
2. The game runs for R rounds or until all the players are kicked.
3. The dealer starts with no money. However, the dealer can have a negative amount of money as she can borrow from the establishment to pay out winners.

3.2.2 Output

The output of the BlackJack function should be a tuple in the following format:
`game(M [R1#M1 R2#M2 … Rn#Mn])`
where `M` is the amount of money the dealer has at the end of the game, `R` is the number of turns that each player stayed in the game and `M` for each player is the amount of money the player finished the game with.

3.3 Extra Credit

For extra credit, you can add two additional features for each player.

3.3.1 Split option

If both the cards in a player's hand have the same rank, the player may choose to split the hand into two separate hands by placing an equal bet on the new hand. For example, consider the case when a player's initial hand is [4Club, 4Diamond] and his bet for the round is 10. When it is his turn to play with the dealer, he can split the hand into two hands [4C] and [4D] by placing additional 10 on the new hand. Your program should also allow a player to split a hand that is already split. In the example, suppose the player is dealt [4Heart] for the [4C] hand then he may choose to split the hand again. He will be playing with three separate hands with a bet of 10 on each hand. The table below provides the strategy for when a player should split:

Player's Hand	Dealer's Open Card
Pair of Aces or 8s	Any
Pair of 2s, 3s, or 7s	Between 2 and 7
Pair of 6s	Between 2 and 6
Pair of 9s	Between 2 and 6 or 8 or 9

3.3.2 Card Counting

Each player in the program should be able to employ high-low card counting strategy to alter the bet amounts after each round. The basic idea of card counting method is to track the count variable through the entire game. At the beginning count is 0. The value of count changes whenever an open card (X) is placed on the table as shown below. It is reset to zero when the cards are reshuffled.

`Count=Count+`	`1`	`2≤Value(X)≤6`
	`0`	`7≤Value(X)≤9`

	`-1`	otherwise

The player in your program should bet twice the normal amount if the value of the count variable is at least +2.

Due Date:

Received Time	**Grade Modification**
before Sunday, 10/05, 11:59PM	+10%
before Monday, 10/06, 11:59PM	no modification (on time)
before Tuesday, 10/07, 11:59PM	-10%
before Thursday, 10/09, 11:59PM	-25%
after Friday, 10/10, 12:00AM	not accepted

Grading: The assignment will be graded mostly on correctness, but code clarity / readability will also be a factor (comment, comment, comment!). See the professor or TAs, if you have ideas for other extensions for this assignment and would like extra credit for implementing them.

Submission Requirements: Please submit a ZIP file with your code, including a README file. In the README file, place the names of each group member (up to two). Your README file should also have a list of specific features / bugs in your solution. Your ZIP file should be named with your LMS user name(s) as the filename, either userid1.zip or userid1_userid2.zip. Only submit one assignment per pair via LMS.