Cmpe 260 Scheme Project

Due Date: April 9, 2017 23:59 PM

(No extension due to midterm!)

1 Introduction

In this project, you are going to implement functions for Mini Card Game. It is a simple playing card game.

2 Mini Card Game

In Mini Card Game, there is only one player. The game is played with a card-list and a goal point. The player has a list of held-cards which is initially empty. The game is played with a list of moves which is provided at the beginning. The player can make 2 types of moves:

`draw :` With this move, the player takes the first card in the card-list and adds it to the held-cards. If the player draws a card c (it must be the first card in the card-list), the length of the card-list is reduced by one and the length of the held-cards is incremented by one.

`discard:` With this move, the player removes any one card from the held-cards. If the player discards a card c, the game continues with the held-cards except c. The card-list is unchanged. The discard card selection strategy is not specified. You can implement your own strategy.

The objective of Mini Card Game is to end the game with a low score.

`Scoring details :` Let `playerpoint` be the sum of the values of held-cards, goal be the goal point which is an integer value. First of all, `prescore` for the player is calculated. The `finalscore` is calculated with `prescore` according to held-cards. The `prescore` is 5 times `( playerpoint - goal )`, if the `playerpoint` is greater than `goal`. Otherwise, the `prescore` is `( goal - playerpoint )`. The `finalscore` is the `prescore`, if there are different colored cards in held-cards list. If all the held-cards are same color, the `finalscore` is the half of the `prescore` (rounded down with integer division).

The game has 4 endings:

`Game-Over-1:` If the list of moves is empty, the game ends.

`Game-Over-2:` If the current move is draw and the card-list is empty, the game ends.

`Game-Over-3:` If the current move is discard and the held-cards is empty, the game ends.

`Game-Over-4:` At any time if the sum of the values in the held-cards is greater than the goal after the last move, the game ends.

3 An Example Play

Game elements are:

1. card-list: It is the deck of cards on the table.
2. move-list: It is the list of moves for the player. It is given at the beginning of the game.
3. goal: It is the minimum score that the player wants to approach.
4. held-cards: It is the list of cards that the player is holding. It is an empty list at the beginning of the game.

Lets play an example game to warm up. Our inputs are:

1. card-list: '((H . 3) (H . 2) (H . A) (D . A) (D . Q) (D . J))
2. move-list: '(draw draw draw discard)
3. goal: 20
4. held-cards: '()

'(H . 3) is three of Hearts and '(D . J) is Jack of Diamonds.

Move 1: draw - Take the first card from card-list and append to held-cards. After this move:
   (a) card-list: '((H . 2) (H . A) (D . A) (D . Q) (D . J))
   (b) move-list: '(draw draw discard)
   (c) goal: 20
   (d) held-cards: '((H . 3))

Move 2: check - The sum of held-cards is 3 and is less than the goal (20). So continue.

Move 3: draw - Take the first card from card-list and append to held-cards. After this move:
   (a) card-list: '((H . A) (D . A) (D . Q) (D . J))
   (b) move-list: '(draw discard)
   (c) goal: 20
   (d) held-cards: '((H . 3) (H . 2))

Move 4: check - The sum of held-cards is 5 and is less than the goal (20). So continue.

Move 5: draw - Take the first card from card-list and append to held-cards. After this move:
   (a) card-list: '((D . A) (D . Q) (D . J))
   (b) move-list: '(discard)
   (c) goal: 20
   (d) held-cards: '((H . 3) (H . 2) (H . A))

Move 6: check - The sum of held-cards is 16 and is less than the goal (20). So continue.

Move 7: discard - Choose one card from held-cards and put it away. For this example lets choose the first card to remove. After this move:
   (a) card-list: '((D . A) (D . Q) (D . J))
   (b) move-list: '()
   (c) goal: 20
   (d) held-cards: '((H . 2) (H . A))

Move 8: check - The sum of held-cards is 13 and is less than the goal (20). So continue.

Move 9: check - The list of moves is empty. So game ends.

Lets calculate our score.

The playerpoint is the sum of held-cards, that is 13 at the end of the game.

The playerpoint is less than the goal, so the prescore is (goal - playerpoint), that is 7.

We have all Hearts in held-cards, therefore all the cards are same color.

The finalscore is half of the prescore because all cards are same color, that is 3 as rounded down.

At the end, our finalscore is 3. Congratulations!

4 The Functions

You should implement the following functions for Mini Card Game:

4.1 (2 points) (card-color one-card)

Returns the color of `one-card`.

Hint: Hearts (H) and Diamonds (D) are red. Spades (S) and Clubs (C) are black.

```
> ( card-color '(H . A) )
'red
> ( card-color '(S . 10) )
'black
```

4.2 (2 points) (card-rank one-card)

Returns the rank of `one-card`.

Hint: Use 11 for ace (A), 10 for king (K) and queen (Q) and jack (J), and numbers

```
> ( card-rank '(H . A) )
11
> ( card-rank '(S . 10) )
10
```

4.3 (6 points) (all-same-color list-of-cards)

Returns `#t` if all the cards in `list-of-cards` have same color, `#f` otherwise.

```
> ( all-same-color '((H . 3) (H . 2) (H . A) (D . A) (D . Q) (D . J)) )
#t
> ( all-same-color '((S . 3) (S . 2) (S . A) (C . A) (C . Q) (C . J))  )
#t
> ( all-same-color '((H . 3) (H . 2) (H . A) (D . A) (D . Q) (C . J))  )
#f
```

4.4 (10 points)
(fdraw list-of-cards held-cards)

Returns a new list of held-cards after the action draw is taken.

```
> ( fdraw '((H . 3) (H . 2) (H . A) (D . A) (D . Q) (D . J)) '())
'((H . 3))
> ( fdraw '((H . 3) (H . 2) (H . A) (D . A) (D . Q) (D . J)) '((S . 3) (S . 2) (S . A)))
'((S . 3) (S . 2) (S . A) (H . 3))
```

4.5 (10 points)
(fdiscard list-of-cards list-of-moves goal held-cards)

Returns a new list of held-cards after the action discard is taken.

Hint: list-of-cards and list-of-moves and goal are also provided as arguments so that you can generate a better gaming strategy than your friends to reach a lower score.

```
> ( fdiscard '((C . 3) (C . 2) (C . A) (S . J) (S . Q) (H . J))
                '(draw draw draw discard) 66
                '((H . 3) (H . 2) (H . A) (D . A) (D . Q) (D . J))        )
'((H . 3) (H . 2) (H . A) (D . A) (D . Q))        ;output
> ( fdiscard '((H . 3) (H . 2) (H . A) (D . A) (D . Q) (D . J))
         '(draw draw draw discard) 56
         '((S . 3) (S . 2) (S . A) (C . A) (C . Q) (C . J))        )
'((S . 3) (S . 2) (C . A) (C . Q) (C . J))        ;output
```

4.6 (15 points)
(find-steps list-of-cards list-of-moves goal)

Returns a list of steps that is a list of pairs of moves and corresponding cards along the game.

Note: Remember and be careful about Game-Over-4!

```
> ( find-steps '((H . 3) (H . 2) (H . A) (D . J) (D . Q) (C . J)) '(draw draw draw discard) 16 )
'((draw (H . 3)) (draw (H . 2)) (draw (H . A)) (discard (H . 3)))
```

4.7 (15 points)
(find-held-cards list-of-steps)

Returns the list of held-cards after the list-of-steps is applied. Remember the list of held-cards is initially empty.

```
> ( find-held-cards '((draw (H . 3)) (draw (H . 2)) (draw (H . A)) (discard (H . 3))) )
'((H . 2) (H . A))
```

4.8 (10 points)
(calc-playerpoint list-of-cards)

Calculates and returns the corresponding `playerpoint` for list-of-cards. The calculation is described in definition part.

Hint: Use 11 for ace, 10 for king and queen and jack, and numbers

```
> ( calc-playerpoint '((H . A) (H . 3) (H . 2) (D . Q) (D . J) (C . J)) )
46
```

4.9 (10 points)
(calc-score list-of-cards goal)

Calculates and returns `finalscore` according to the definition part using inputs `list-of-cards` and `goal`.

```
> ( calc-score '((H . 3) (H . 2) (H . A) (D . J) (D . Q) (C . J)) 50 )
4
> ( calc-score '((H . 3) (H . 2) (H . A) (D . J) (D . Q) (C . J)) 16 )
150
```

4.10 (20 points)
(play list-of-cards list-of-moves goal)

Returns `finalscore` at the end of the game after processing (some or all of) the moves in the move list in order.

```
> ( play '((H . 3) (H . 2) (H . A) (D . J) (D . Q) (C . J)) '(draw draw draw discard) 16 )
1
    (Note: (H . 3) is discarded for discard move.)
```

5 BONUS (10 points)

We are going to test your `play` functions in a competition. You will be able to receive bonus points according to your rank in the competition. Remember lowest score wins!

6 Documentation and Clarity

Documenting the code is essential for developing in the long term and as you're not proficient in functional programming very much, you tend to solve the problems in rather obscure and/or unnatural ways. Thus, you should document every predicate in your project and you will be graded for documentation of your code in case your code becomes hard to understand.

You should add documentation to each function you defined (both the ones required by the project description ad the ones you've created as helpers) in the form below (which is sort of the de facto documentation format for Racket):

```
; (hypot x y) -> number?
; x : number?
; y : number?
;
; Calculates hypotenuse of the right triangle formed by x and y.
;
; Examples:
; > (hypot 3 4)
; => 5
; > (hypot 5 12)
; => 13
(define (hypot x y)
  (sqrt (+ (* x x) (* y y))))
```

Here first line is function's signature, second and third lines are parameter types, fifth line is function description and we have a few examples on lines 8-11.

Besides documentation, it is also important to write code that is readable, so avoid small, clever, hard-to-comprehend hacks (unless they fit the very nature of functional programming, in which case you should explain the hack in documentation) and unnecessary complexity in your code. You are also graded for code clarity. When you cannot find a clearer way to handle something, explain how your code works in documentation, as it is important for us to grade your project and to be sure that you understood the concepts behind functional programming.

7 Submission

You are going to submit a file explicitly named as `YOUR_STUDENT_ID.rkt` including the filename extension, all lowercase. The first 4 lines of the code must be in the form:

```
#lang scheme
; compiling: yes
; complete: yes
; student id
```

The first line declares language as Scheme. The second line denotes whether your code is compiled correctly[1], the third line denotes whether you completed all of the project, which must be "no" (without quotes) if you're doing a partial submission. `student id` must match with your student ID. The four lines must be lowercase.

Check that you're submitting the code as described above. Explicitly check that,

1. The filename is correct, including the extension.
2. You submitted the right file, you can check that by downloading your submission from Moodle.
3. You didn't submit a zip file or something else.
4. The filename is all in *lowercase*.

[1] here to be compiled correctly means that you get no errors when you press the Run button to run your code

8 Prohibited Constructs

The following language constructs are ***explicitly prohibited***. You *will not get any points* if you use them:

1. Any function or language element that ends with an `!`.
2. Any of these constructs: `begin`, `begin0`, `when`, `unless`, `for`, `for*`, `do`, `set!-values`.
3. Any language construct that starts with `for/` or `for*/`.
4. Any construct that causes any form of mutation[2] (impurity).

9 Some Tips on The Project

- You can check out PS slides.
- You can (and should) use higher-order functions like `map`, `filter`, `foldl` and `foldr`. You are also encouraged to use anonymous functions with help of `lambda`. The functions in project are designed to make use of them.
- You may need to define your own improvements (such as defining an extended `cons`) to write some functions more easily.
- You can use Racket reference, either from DrRacket's menu: Help > Racket Documentation, or from the following link: http://docs.racket-lang.org/reference/index.html
- Simply Scheme: Introducing Computer Science is a nice book for learning Scheme: https://people.eecs.berkeley.edu/~bh/ss-toc2.html
- SICP (Structure and Interpretation of Computer Programs a.k.a. The Purple Book) is a nice book for learning Scheme: https://mitpress.mit.edu/sicp/full-text/book/book.html
- There are also video lectures using purple book on MIT OCW, by the authors: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-001-structure-and-interpretation-of-computer-programs-spring-video-lectures/

10 Appendix ♡♣♢♠

In a standard deck of playing cards, there are 52 cards.

Each card has color, suit and rank properties.

There are two possible colors: black and red

There are four suits: Hearts (♡), Clubs (♣), Diamonds (♢), Spades (♠)

Hearts and Diamonds are red. Spades and Clubs are black.

The ranks are ace (A), king (K), queen (Q), jack (J) and numbers from 10 down to 2.

Each suit has one card for each rank.

[2]Mutation means creating any kind of change, either via changing value of a variable, a memory slot, or an element of a container; or via doing input/output.