Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Lenguajes de Programación
Proyecto 2

Jugando cartas con un funcional – Parte A

Este proyecto se va a dividir en 2 partes. La parte A que deberá ser desarrollada como complemento a la parte B (recuerden hacer 2 versiones del código). Y la parte B que debe ser entregada el 6 de abril (la parte B es más pequeña). A continuación se explicará en detalle la parte A de este proyecto.

(1) Vamos a crear un juego de cartas ficticio para este proyecto. Este juego se llama "**card challenge**". Este juego se lleva a cabo con un *card-list* y un *goal*. Un jugador tiene una lista de *held-cards*, inicialmente *empty*. El jugador lleva a cabo una movida haciendo uno de los dos siguientes movimientos: un *drawing*, que significa remover la primera carta en el *card-list* y añadiendo esta carta en el held-cards. El otro movimiento es hacer un *discarding*, que significa elegir una carta de las *held-cards* para remover. El juego finaliza cuando un jugador elige no hacer más movimientos o cuando la suma de los valores del *held-cards* es mayor que el *goal*.

Elementos involucrados:

- card-list: lista.
- held-cards: lista.
- goal: número que no hay que sobrepasar.
- empty: indicador de lista vacía.
- drawing: sacar una carta del card-list y agregarla al held-cards.
- discarding: elegir una carta del held-cards para remover.

(2) El objetivo es finalizar el juego con un *score* bajo (0 sería el mejor). Y este score funciona de la siguiente manera: sea *sum* la suma de los valores de held-cards. Si *sum* es mayor que *goal*, entonces el *preliminary score* es tres veces (*sum* – *goal*), si no, entonces el *preliminay score* es (*goal* – *sum*). Finalmente, se puede decir que *score* (el resultado final) es el *preliminary score*, a menos de que todas las cartas en *held-cards* sean del mismo color, en ese caso, el *score* sería el *preliminary score* dividido entre 2 (con redondeo de piso).

Elementos involucrados:

- preliminary score: cálculo preliminar del score.
- score: resultado final del juego.

(3) Funciones a desarrollar**:**

(a) Escriba una función **card_color**, la cual toma una carta y retorna su color (spades and clubs son negras, diamonds and hearts son rojas).

(b) Escriba una función **card_value**, la cual toma una carta y retorna su valor (las

cartas numeradas tiene su número como valor, los aces valen 11, todas las demás valen 10).

(c) Escriba una función **remove_card**, la cual toma una lista de cartas **cs**, una carta **c**, y una excepción **e**. Retorna una lista que tiene todos los elementos de **cs** excepto **c**. Si **c** está más de una vez, se debe remover solamente la primera **c** que encuentra. Si **c** no está en la lista, entonces se debe lanzar la excepción **e**.

(d) Escriba una función **all_same_color**, la cual toma una lista de cartas y retorna **true** si todas las cartas que están dentro de la lista, son del mismo color. (Nota: usen pattern maching anidado como lo vimos en clase).

(e) Escriba una función **sum_cards**, la cual toma una lista de cartas y retorna la suma de sus valores. (Nota: para este problema es requerido que la solución utilice la técnica *tail recursion*).

(f) Escriba una función **score**, que toma una lista de cartas (en este caso serían las *held-cards*) y un **int** (en este caso sería el *goal*) y determina el score basado en el apartado **(2)** de este enunciado.

(g) Escriba una función **officiate**, la va a "correr el programa". Esta función toma la lista de cartas (*card-list*), una lista de movimientos conocida como *move-list* (lo que el jugador haría en cada punto), y un **int** (que sería el *goal*); y al final va a retornar el score obtenido al final del juego obtenido después de procesar los movimientos (o algunos de ellos) contenidos en el *move-list* (en orden secuencial). Nota: use una *helper function* que toma varios argumentos que en conjunto representan el estado del programa. A continuación se describe parte del programa:

(i) El juego inicia en la *held-cards* estando vacía.

(ii) El juego termina si ya no hay más movimientos. El jugador elige parar si la *move-list* es *empty*.

(iii) Si el jugador decide hacer un *discard* de alguna carta **c**, el juego continúa, en ese caso con la *held-cards* sin **c** y con la *card-list* sin cambios. Si **c** no está en las *held-cards,* lance la excepción **IllegalMoveException**.

(iv) Si el jugador decide hacer un movimiento de *drawing*, y la *card-list* es *empty*, el juego termina. En cambio, si el movimiento de *drawing* causa que la suma de las *held-cards* exceda el *goal*, el juego termina (después del *drawing*). Si no suceden las condiciones anteriores, entonces el juego continúa con una *held-cards* más grande y una *card-list* más pequeña.

(v) Para tomarlo como una guía: officiate debería de tomar cerca de 20 líneas de código.

(4) Código de referencia:

Inicie programando con el siguiente código como referencia (debe estar al inicio de la solución):

```
datatype suit = Clubs | Diamonds | Hearts | Spades
datatype rank = Jack | Queen | King | Ace | Num of int
type card = suit * rank

datatype color = Red | Black
datatype move = Discard of card | Draw

exception IllegalMove
```

(5) Para desarrollar de manera adecuada las funciones, a continuación se listan los nombres de las funciones y sus respectivos tipos, para que puedan desarrollar los métodos de manera adecuada y también para hacer pruebas:

- *val card_color = fn : card -> color*
- *val card_value = fn : card -> int*
- *val remove_card = fn : card list * card * exn -> card list*
- *val all_same_color = fn : card list -> bool*
- *val sum_cards = fn : card list -> int*
- *val score = fn : card list * int -> int*
- *val officiate = fn : card list * move list * int -> int*

(6) Se proporcinonan algunas pruebas preliminares para verificar la solución que vayan desarrollando:

```
val test1 = card_color (Clubs, Num 2) = Black

val test2 = card_value (Clubs, Num 2) = 2

val test3 = remove_card ([(Hearts, Ace)], (Hearts, Ace), IllegalMove) = []

val test4 = all_same_color [(Hearts, Ace), (Hearts, Ace)] = true

val test5 = sum_cards [(Clubs, Num 2),(Clubs, Num 2)] = 4

val test6 = score ([(Hearts, Num 2),(Clubs, Num 4)],10) = 4

val test7 = officiate ([(Hearts, Num 2),(Clubs, Num 4)],[Draw], 15) = 6

val test8 = officiate ([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)],
                       [Draw,Draw,Draw,Draw,Draw],
                       42)
          = 3
```

Notas finales

- Este proyecto debe ser resuelto en parejas. Pero hay una condición: ambas personas deben dominar el código, de lo contrario habrá una rebaja de puntos.
- Crear un solo archivo con la solución de la tarea.
- Hacer los casos de prueba.
- No se puede hacer uso de null, hd, tl y #.
- Fecha de entrega: _____.