Lenguajes de Programación
Tarea N°1

Entrega: Miércoles 25 de Marzo

Consulte las normas de entrega de tareas en `http://pleiad.cl/teaching/cc4101`.

1 (2.0) Programación Funcional

1. (0.6) Implemente la función `zipWith :: (Listof A) (Listof B) (A B → C) → (Listof C)` que "pega" dos listas, utilizando una función que combina los elementos de ambas listas. Las listas no necesariamente deben ser del mismo largo; en este caso sólo se combinan los elementos hasta completar la lista más pequeña. Ejemplo:

```
> (zipWith '(1 2 3) '(1 2 3) equal?)
  '(#t #t #t)

> (zipWith '(1 3) '(2 4 6) +)
  '(3 7)
```

2. (0.2) Implemente, utilizando `zipWith`, la función `zip :: (Listof A) (Listof B) → (Listof (A ,B))` que combina las listas en una sola, obteniendo una lista de pares tomando elementos de la misma posición. Ejemplo:

```
> (zip '(1 2 3) '(3 2 1))
  '((1 . 3) (2 . 2) (3 . 1))

> (zip '(8 9 10 11 13) '(11 10 9))
  '((8 . 11) (9 . 10) (10 . 9))
```

3. (0.8) Utilizando (al menos) `zipWith`, `foldr` y `map` genere una función `eq-fun :: (Num → A) (Num → A) Num Num → Num` que determine la cantidad de resultados iguales que otorgan dos funciones `f1` y `f2` para una lista de inputs entre `i` y `j` (considerando ambos valores). Ejemplo:

```
> (eq-fun odd? odd? 0 10)
  11

> (eq-fun odd? even? 4 8)
  0
```

4. (0.4) Utilizando `foldr` defina la función `sep-list :: (Listof A) → (Listof A)` que separa en sublistas todos los elementos consecutivos que sean iguales. Ejemplo:

```
> (sep-list '(3 1 5 6 1 1 1 3 4))
  '((3) (1) (5) (6) (1 1 1) (3) (4))

> (sep-list '(#t #t #f #f #t #t))
  '((#t #t) (#f #f) (#t #t))
```

2 (4.0) Face-Check Cards

Considere un mazo de 52 cartas. El objetivo es obtener 9 cartas pertenicientes al mazo, a la cual llamaremos mano, con el menor puntaje posible. El juego es dividido en rondas. Cada ronda requiere una cantidad de trios de cartas (3 cartas de un cierto número sin importar su pinta) y una cantidad de escalas (4 cartas consecutivas de la misma pinta sin incluir As intermedios ni al final de la escala) que varian dependiendo del tipo de ronda, para calcular el puntaje final de la mano.

Cada carta tiene un puntaje equivalente a su número. Consideraremos las siguientes excepciones: la carta Jack, Queen y King tendrán como número 11, 12, 13 respectivamente con un puntaje asignado de 10 cada una y la carta As tendrá asignado un 1 como número pero su puntaje será de 15. No consideraremos la carta Jocker.

Para implementar las funciones que representarán las rondas y calcularán el puntaje necesitamos definir dos estructuras. En primer lugar definimos `Card`:

```
(deftype Card
  (spades number)
  (hearts number)
  (diamonds number)
  (clubs number))
```

donde cada elemento representa una carta, especificando su pinta (spades, hearts, diamonds o clubs) con su respectivo número.

También definimos `Meld` de la siguiente forma:

```
(deftype Meld
  (str rank fst)
  (toak num))
```

donde `(toak num)` (`toak` entendida como sigla de Three-Of-A-Kind) representa un trio de tres cartas de número `num` y `(str rank fst)` (`str` entendida como abreviación de straight) representa una escala de 4 cartas consecutivas de la pinta, representada con el string `rank`, cuyo primer número es `fst`. Note que ambos tipos de uniones (trios y escalas) son excluyentes: Una misma carta no puede estar en un trio y una escala a la vez para una misma mano.

El puntaje de una mano en cada ronda es calculado de la siguiente forma: Se descartan de la mano todas las cartas que pertenecen al tipo de ronda y se suma el puntaje del resto de las cartas. En caso de que la mano no cumpla con la cantidad de trios y/o escalas pedidos en la ronda, se suman todas las cartas de la mano.

Para el caso de funciones que definen rondas, si la mano no cumplió con la cantidad de trios y/o escalas esperadas, la función que debe retornar una lista vacía. En la lista de `Meld` que retorna cada una de las funciones que definen una ronda siempre las escalas preceden a los trios y entre ellos se preceden de mayor a menor puntaje. En caso de que los puntajes sean iguales se considera también la siguiente precedencia de pintas para las escalas desde mayor a menor precedencia: spades → hearts → diamonds → clubs.

1. Implemente en Racket las siguientes funciones que definen una ronda:

1.1. (0.3) `two-toak :: (Listof Cards) → (Listof Meld)`: Verifica que hay (al menos) dos trios de cartas del mismo número y los retorna. Si existen más trios en una mano de los que la función requiere, se deben elegir los trios que sumen más puntaje. Ejemplo:

```
> (two-toak (list (spades 1) (hearts 1) (diamonds 1)
                  (clubs 3) (hearts 3) (diamonds 3)
                  (clubs 2) (hearts 2) (diamonds 2)))

  (list (toak 1) (toak 3))
```

1.2. (0.6) `one-str-one-toak cards :: (Listof Cards) → (Listof Meld)`: Verifica que hay (al menos) una escala y un trio y los retorna. Se aplica la misma lógica anterior para las escalas:

```
> (one-str-one-toak (list (clubs 10) (clubs 2) (diamonds 2)
                          (clubs 2) (hearts 4) (diamonds 4)
                          (clubs 7) (clubs 8) (clubs 9)))

  (list (str "clubs" 7) (str 2))
```

1.3. (0.4) `two-str cards :: (Listof Cards) → (Listof Meld)`: Verifica que hay (al menos) dos escalas de distinta pinta.

1.4. (0.3) `three-toak cards :: (Listof Cards) → (Listof Meld)`: Verifica

que hay tres trios.

2. (0.3) Utilizando `foldr`, defina `ranked-hand? :: (Listof Cards) → Bool` que determina si todas las cartas de la mano son de la misma pinta.

3. (0.3) Utilizando `foldr`, defina `straight-hand? :: (Listof Cards) → Bool` que determina si todas las cartas de la mano forman una escala de 9 cartas sin importar su pinta.

4. (0.6) Defina la función `puntaje-ronda :: ((Listof Cards) → (Listof Meld)) (Listof Cards) → Num` que calcula el puntaje de una ronda.

5. (1.2) Defina la función `best-play :: (Listof Cards → String)` que retorna un string con el nombre de la función que otorga menor puntaje a la mano luego de haber aplicado la ronda. En caso de que existan dos, se retorna la que necesite menor cantidad de cartas. Ejemplo:

```
> (best-play (list (clubs 1) (hearts 1) (diamonds 1)
                   (clubs 4) (hearts 4) (spades 4)
                   (spades 5) (spades 6) (spades 7)))

  "one-str-one-toak"
```