Simulacro de primer parcial

Laboratorio de AYEDI - 2C2024

> *Además del formulario que deberás completar con las respuestas, debes entregar el código completo. Este código debe poder ejecutarse en haskell sin errores. Te recomendamos para ello que pruebes con diferentes ejemplos antes de entregar.*

Descripción del problema

Queremos definir un tipo de datos para representar a los deportistas que participan en carreras. Los deportistas pueden ser velocistas o ciclistas, y deben modelarse utilizando un tipo de datos que capture sus características principales.

Ejercicio 1

a)

Definir el tipo `Deportista` que consta de dos constructores `Velocista` y `Ciclista` con los siguientes parámetros:

- El constructor `Velocista` debe tomar como parámetro un número entero que represente la altura de la persona en centímetros.
- El constructor `Ciclista` debe tomar como parámetro un valor de tipo `Modalidad`, que puede ser `Carretera` o `Pista`.

b)

A partir del tipo definido en el punto anterior, definí el valor `juan` que representa a un deportista velocista con una altura de 172 cm.

```
juan :: Deportista
juan = -- COMPLETAR
```

c)

Definir la función **`esVelocistaAlto`** `:: Deportista -> Int -> Bool` que dado un deportista y un entero `n`, devuelve **`True`** si el deportista es un velocista y su altura es mayor a `n`. De lo contrario, devuelve **`False`**.

d)

Programá la función `contarVelocistas :: [Deportista] -> Int` que dada una lista de deportistas `xs`, devuelve la cantidad de velocistas en la lista.

e)

Definir la función `esCiclista :: Deportista -> Bool` que, dado un deportista, devuelve `True` si el deportista es ciclista y `False` en caso contrario.

Ejercicio 2

Supongamos que queremos representar una cola de deportistas, como aquellas que forman fila para retirar sus credenciales en la villa olímpica. Un deportista llega y se coloca al final de la cola y espera su turno. El orden de atención respeta el orden de llegada, es decir, quien llega primero es atendido primero. Podemos representar esta situación con el siguiente tipo de datos:

```
data Cola = VaciaC | Encolada Deportista Cola deriving Show
```

a)

Definir la función `encolar :: Deportista -> Cola -> Cola`, que toma un deportista y una cola de deportistas, y devuelve una nueva cola con el deportista agregado en la última posición.