Parcial – Estructuras de Datos – UNQ

Aclaraciones:

- *Esta evaluación es a libro abierto. Se pueden usar todas las funciones y tipos de datos vistos en la práctica y en la teórica, salvo que el enunciado indique lo contrario.*
- *No se olvide de poner nombre, nro. de alumno, nro. de hoja y cantidad total de hojas en cada una de las hojas.*
- *Le recomendamos leer el enunciado en su totalidad y organizar sus ideas antes de comenzar la resolución.*
- *Recuerde que la intención es medir cuánto comprende usted del tema. Por ello, no dude en escribir todo lo que sabe, en explicar lo que se propone antes de escribir código, en probar sus funciones con ejemplos, etc.*

1. Introducción

Por los pasillos de la universidad se comenta que en Estructuras de Datos se suele tomar mate a menudo. Como alumno de este curso en instancia de parcial, usted debe demostrar que cuenta con los conocimientos de programación de estructuras de datos que forman parte de los contenidos de la materia, y para esto deberá desarrollar la lógica de un sistema que administra rondas de mate en un momento dado. Este sistema se compone de distintos tipos de datos, que serán descritos a lo largo de este parcial. Para resolver esta tarea empleará el lenguaje de programación Haskell.

Cabe mencionar que cada tipo abstracto de datos debe encontrarse en su propio módulo, y que debe definir entonces el encabezado de cada módulo e importar otros según corresponda. Recuerde que queremos evaluar sus conocimientos y por lo tanto no podrá utilizar funciones que haya programado durante la cursada (deberá redefinirlas en esta instancia de parcial si las necesita). Sin embargo, sí podrá hacer uso de otros tipos abstractos (mediante sus interfaces) que utilizaremos para implementar distintas estructuras nuevas.

2. Implementación

Ejercicio 1 Primero se pide implementar el tipo abstracto `Termo`, representado simplemente como una cantidad de mates que permite tomar:

```
data Termo = T int
```

Un termo se encuentra lleno cuando permite tomar 20 mates, y vacío cuando ya no permite tomar ningún mate. La interfaz de este tipo de datos es la siguiente:

- `termoLleno :: Termo`
  Crea un termo que permite tomar 20 mates.
- `estaVacio :: Termo -> Bool`
  Recibe un termo e indica si está vacío.
- `tomarMate :: Termo -> Termo`
  Recibe un termo y decrementa en uno la cantidad de mates que permite tomar. Esta función es parcial cuando el termo se encuentra vacío.

Implemente el tipo abstracto `Termo` utilizando la representación presentada anteriormente, e indicando invariantes de representación correspondientes.

Ejercicio 2 Como describimos en la introducción, este sistema administra rondas. En este punto implementaremos el tipo abstracto `Ronda`, que se compondrá de un termo, un gusto (amargo o dulce) y una cola con los DNI de las personas que se encuentran tomando mate en esa ronda. Las rondas no admiten más de 10 personas (si posee 10 personas la ronda se considera llena). Su representación se implementa de la siguiente manera:

```
data Ronda = R Termo Gusto (Queue DNI)
data Gusto = Amargo | Dulce
type DNI = Int
```

La interfaz de este tipo de datos es la siguiente:

- `rondaVacia :: Gusto -> Ronda`
  Crea una ronda con un termo lleno, del gusto indicado por parámetro y sin personas en la ronda.
- `gustoDeRonda:: Ronda -> Gusto`
  Devuelve cuál es el gusto de la ronda.
- `cantidadDeGente :: Ronda -> Int`
  Indica cuántas personas posee la ronda.
- `agregarDni:: DNI -> Ronda -> Ronda`
  Encola una persona a la ronda. Esta función es parcial si la persona ya se encuentra en la ronda, o si la ronda se encuentra llena.
- `cebarMate:: Ronda -> Ronda`
  Le ceba un mate a la siguiente persona en la ronda, y recarga el mate si se encuentra vacío luego de cebar. Esta función es parcial si no hay personas en la ronda.
- `cebarAtodos:: Ronda -> Ronda`
  Ceba un mate a cada persona en la ronda, recargando el termo al vaciarse hasta completar la ronda.

Implemente este tipo abstracto utilizando la representación y la interfaz descritas, indicando los invariantes de representación correspondientes.

Ejercicio 3 Finalmente implementaremos el tipo abstracto `Curso`, que se representa como una lista de rondas:

```
data Curso = C [Ronda]
```

La interfaz del tipo `Curso` es la siguiente:

- `esCursoPerfecto :: Curso -> Bool`
  Dado un curso indica si posee al menos una ronda con cada gusto posible.
- `asignarleRonda :: Gusto -> Dni -> Curso -> Curso`
  Toma un gusto, un DNI y un curso, y agrega la persona a la primera ronda de la lista del curso que posea ese gusto y no se encuentre llena. Si no se encuentra una ronda así, crea una nueva con el gusto y la persona dados.
- `crearCurso :: Map Dni Gusto -> Curso`
  Dado un Map que relaciona DNI y gustos, crea un curso con esta información (las personas pueden agregarse en cualquier orden al curso).

Implementar el tipo `Curso` con la representación e interfaz descritas, indicando los invariantes de representación que considere necesarios.

3. Ejercicio de penalización por inasistencias

Ejercicio 4 Ampliaremos el tipo abstracto `Ronda` para contener un paquete de bizcochos:

```
data Ronda = R Termo Gusto (Queue DNI) PaqueteBizcochos

data PaqueteBizcochos = Paq Cant (Maybe Cupon)
type Cant = Int

data Cupon = SigaParticipando
           | Ganaste PaqueteBizcochos
           | DoblePremio PaqueteBizcochos PaqueteBizchochos
```

Junto a esta modificación se amplía la interfaz con las siguientes funciones:

- `cantidadDePremios :: Ronda -> Int`
  Indica la cantidad de premios en total que se ganaron por el paquete de bizcochos de la ronda (o sea, los premios de ese, y los premios de los premios, etc.)
- `bizcochosPorPersona :: Ronda -> Int`
  Indica cuántos bizcochos se deben repartir a cada persona de la ronda; el total de bizcochos se reparte en igual cantidad para todos los participantes (y los que sobran no se usan).