UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2012

04 de mayo de 2012

TPF - OJOTA

1. Introducción

En el TPE, hemos especificado el comportamiento de una serie de problemas relacionados con los juegos olímpicos. Nuestro próximo paso será obtener un programa funcional que respete la especificación que adjunta la cátedra para esta nueva etapa del proyecto. El objetivo de este trabajo es, entonces, programar en Haskell los tipos y operaciones del TPE. A continuación diremos cómo decidimos implementar cada tipo y qué funciones exporta (interfaz). La interfaz de los tipos no podrá ser modificada, salvo por expresa (expresa==escrito) indicación de la cátedra. En cada tipo pueden implementar su propio `show`, de tal manera que se sientan cómodos con su visualización. **NO** pueden agregar el `Eq` si el mismo no se encuentra en el presente enunciado. Se pueden agregar funciones auxiliares dentro de cada módulo, pero **NO** se pueden exportar.

Para la resolución del trabajo, se pueden usar únicamente las funciones y operadores `last`, `init`, `head`, `tail`, `!!`, `reverse`, `++`, `elem`, `length`, `fst`, `snd` y los operadores de comparación entre elementos de un mismo tipo. Notar que la especificación que se adjunta **NO** es una solución al tp de especificación.

Los tipos **NO** pueden ser transformados en listas para la implementación de los problemas.

Su trabajo consistirá en implementar todos los problemas especificados por la cátedra (ver archivo aparte). A continuación se detalla la definición de los módulos que deben implementar.

2. Módulo Tipos

```
module Tipos where

type Deporte = String
type Pais = String
type Categoria = (Deporte, Sexo)

data Sexo = Femenino | Masculino deriving (Show, Eq)
```

3. Módulo Atleta

```
module Atleta (Atleta, nuevoA, nombreA, sexoA, anioNacimientoA,
                nacionalidadA, ciaNumberA, deportesA, capacidadA,
                entrenarDeporteA)
where

import Tipos

data Atleta =   A String Sexo Int Pais Int [(Deporte, Int)]  deriving (Show)
```

donde el constructor `A` recibe los parámetros en el siguiente orden:

1. nombre del atleta
2. sexo
3. año de nacimiento
4. nacionalidad
5. cia number
6. lista de pares ordenados, correspondiente a deporte y su correspondiente capacidad. Esta lista siempre debe estar ordenada utilizando como criterio el deporte.

4. Módulo Competencia

```
module Competencia (Competencia, nuevaC, categoriaC, participantesC,
                    finalizadaC, rankingC, lesTocoControlAntiDopingC,
                    leDioPositivoC, finalizarC, linfordChristieC,
                    gananLosMasCapacesC, sancionarTrampososC)

where

import Tipos
import Atleta

data Competencia =    C Categoria
                    | Participar Atleta Competencia
                    | Finalizar [Int] [(Int, Bool)] Competencia
                    deriving (Show)
```

donde:

- el constructor `C` permite crear una competencia con la categoría asignada y sin atletas.
- el constructor `Participar` permite agregar un atleta a una competencia.
- el constructor `Finalizar` permite finalizar una competencia y los parámetros representan (en el mismo orden):
  - la lista de *ciaNumber* de los atletas de acuerdo al orden que tendrán en el ranking.
  - la lista de pares cuya primer componente corresponde al *ciaNumber* del atleta que participó del control anti doping, y la segunda componente es el resultado de dicho control.
  - la competencia a finalizar.

En nuestra representación no vamos a permitir agregar un participante luego de que la competencia ha sido finalizada. Es decir, ninguna instancia de Competencia que se use como parámetro de Participar puede estar construida usando el constructor Finalizar.

5. Módulo JJOO

```
module JJOO (JJOO, nuevoJ, anioJ, atletasJ, cantDiasJ, cronogramaJ,
             jornadaActualJ, dePaseoJ, medalleroJ,
             boicotPorDisciplinaJ, losMasFracasadosJ, liuSongJ,
             stevenBradburyJ, uyOrdenadoAsiHayUnPatronJ, sequiaOlimpicaJ,
             transcurrirDiaJ)

where

import Tipos
import Atleta
import Competencia

data JJOO =     J Int [Atleta] Int
            |   NuevoDia [Competencia] JJOO
            deriving (Show)
```

donde:

- el constructor `J` permite crear una instancia de Juegos Olímpicos y los parámetros representan (en el mismo orden):
  - el año en que se desarolla
  - los atletas que participan
  - el día actual
- el constructor `NuevoDia` permite representar todas las competencias desarrolladas ese día. La primera vez que se utiliza este constructor, refiere al día 1, la segunda vez al día 2 y así sucesivamente.

La cantidad total de días de los JJOO se puede calcular a partir de la cantidad de veces que se aplicó el constructor NuevoDia. Por ejemplo, si se aplicó 3 veces, entonces los JJOO constan de 3 días.

En nuestra representación, en el caso particular que el parámetro día actual sea mayor a la cantidad de días, vamos a interpretar que día actual es exactamente la cantidad de días, ignorando su valor original.