UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructura de Datos I
Parcial de Programación Funcional– 22/05/2009

LU:
Apellidos:
Nombres:

Aclaraciones: El parcial NO es a libro abierto. Cualquier decisión de interpretación que se tome debe ser aclarada y justificada. Para aprobar se requieren al menos 60 puntos. Entregar cada ejercicio en hoja separada. No está permitido utilizar alto orden.

Contamos el tipo abstracto `Buscaminas`, definido de la siguiente manera:

```
tipo Buscaminas {
  observador cantidadFilas (b: Buscaminas) : ℤ;
  observador cantidadColumnas (b: Buscaminas) : ℤ;
  observador hayBomba (b: Buscaminas, f: ℤ, c: ℤ) : Bool;
      requiere 1 ≤ f ≤ cantidadFilas(b) ∧ 1 ≤ c ≤ cantidadColumnas(B);
  observador cantidadBombasAlrededor (b: Buscaminas, f: ℤ, c: ℤ) : ℤ;
      requiere 1 ≤ f ≤ cantidadFilas(b) ∧ 1 ≤ c ≤ cantidadColumnas(b ∧ ¬hayBomba(b, f, c);
}
```

Las funciones que implementan a este tipo en Haskell son

```
cantF::Buscaminas->Int
cantC::Buscaminas->Int
hayBomba::Buscaminas->Int->Int->Bool
cantBA::Buscaminas->Int->Int->Int
```

Ejercicio 1. [45 puntos] Implementar en Haskell los siguientes problemas especificados más adelante

a) **[20 p.]** `problema estaBienFormado` (b : Buscaminas) = `result` : Bool
b) **[15 p.]** `problema cruzando` (b : Buscaminas, c: [(ℤ, ℤ)]) = `result` : Bool
c) **[10 p.]** `problema caminoMasCortoQueCruza` (b : Buscaminas, cs: [[(ℤ, ℤ)]]) = `result` : [(ℤ, ℤ)]

```
problema estaBienFormado (b : Buscaminas) = result : Bool {
  asegura (∀f ← [1..cantidadFilas(b)], c ← [1..cantidadColumnas(b)], ¬hayBomba(b, f, c))
    cantidadBombasAlrededor(b, f, c) == acum(x + 1 | x : ℤ = 0, i ← [f − 1..f + 1], j ← [c − 1..c + 1],
    1 ≤ i ≤ cantidadFilas(b) ∧ 1 ≤ j ≤ cantidadColumnas(b), hayBomba(b, i, j));
}

problema cruzando (b : Buscaminas, c: [(ℤ, ℤ)]) = result : Bool {
  asegura result == loCruza(b, c);
}

problema caminoMasCortoQueCruza (b : Buscaminas, cs: [[(ℤ, ℤ)]]) = result : [(ℤ, ℤ)] {
  asegura loCruza(b, result);
  asegura result ∈ cs;
  asegura (∀k ← cs, loCruza(b, k))|result| ≤ |k|;
}

aux sonContiguas (a,b: [(ℤ, ℤ)]) : Bool = −1 ≤ prm(b) − prm(a) ≤ 1 ∧ −1 ≤ sgd(b) − sgd(a) ≤ 1;
aux loCruza (b : Buscaminas, c: [(ℤ, ℤ)]) : Bool = (∀x ← c)
  prm(x) ∈ [1..cantidadFilas(b)] ∧ sgd(x) ∈ [1..cantidadColumnas(b)] ∧ ¬hayBomba(b, prm(x), sgd(x)) ∧
  (∀i ← [0..|c| − 1))sonContiguas(c[i], c[i + 1]) ∧ c[0] == [(1, 1)] ∧ sgd(c[|c| − 1]) == cantidadColumnas(b);
```

Tipos algebraicos.

Nota Importante: No está permitido utilizar el tipo lista para resolver los ejercicios de tipos algebraicos (Ejercicio 2).

Se cuenta con el tipo compuesto Examen que modela un examen. Básicamente, sobre una hoja en blanco, se pueden ir agregando encabezados o ejercicios con sus respectivos puntajes.

Se busca implementar en Haskell algunos problemas sobre el tipo compuesto Examen mediante el tipo algebraico `Examen`. Dicho tipo está definido con los siguientes constructores

```
data Examen = HojaEnBlanco | Ejercicio Int Examen | Encabezado Examen
```

que permiten crear un Examen "en blanco" (HojaEnBlanco), agregar un ejercicio y su puntaje, y un encabezado respectivamente. Por ejemplo, un examen con 2 encabezados y 2 ejercicios, se puede construir de la siguiente manera:

```
Encabezado (Ejercicio 30 (Ejercicio 5 (Encabezado (HojaEnBlanco))))
```

Ejercicio 2. [**15 puntos**] Implementar `encabezar::Examen->Examen`, que dado un examen, devuelve otro examen que resulta de ordenar el primero de la siguiente manera: a) antes que cualquier ejercicio, deben aparecer todos los encabezados en el mismo orden que se encuentran y b) los ejercicios deben aparecer en el mismo orden que el examen original.

Ejemplo:

```
Ejercicio 10 (Encabezado (Ejercicio 20 (Ejercicio 5 (Encabezado (HojaEnBlanco)))))
```

debería quedar:

```
Ejercicio 10 (Ejercicio 20 (Ejercicio 5 (Encabezado (Encabezado (HojaEnBlanco)))))
```

Ejercicio 3. [**15 puntos**] Implementar `estaBienFormado::Examen->Bool`, que dado un examen, devuelve verdadero sí y sólo si: a) antes que cualquier ejercicio, aparecen todos los encabezados y b) los ejercicios aparecen en orden según su puntaje asociado (primero los de mayor puntaje).

Ejercicio 4. [**25 puntos**] (Ejercicio tomado de la práctica 7) *descomponerEnPrimos* :: $[Int] \rightarrow [[Int]]$, que devuelve la lista de listas, que resulta de descomponer en números primos cada uno de los números de la lista original. Por ejemplo *descomponerEnPrimos* $[2, 12, 6]$ es $[[2], [2, 2, 3], [2, 3]]$. Nota: no importa el orden de cada lista de primos.