Parcial Llenguatges de Programació

Grau en Enginyeria Informàtica

6 Maig 2013

Per accedir al racó aneu a `https://examens.fib.upc.es`
Cal que lliureu via racó el codi amb els comentaris que considereu necessaris en un arxiu ".hs" executable en l'entorn ghci i que satisfaci els diferents enunciats que es llisten a continuació.

Cal que al començar la solució de cada problema afegiu una línia comentada indicant el problema (i subapartat, si en té) que ve a continuació. Per exemple,

```
-- Problema 1.a
-- Problema 2
```

Es valorarà l'ús que es faci de funcions d'ordre superior predefinides. Ara bé, en principi només s'han d'usar les de l'entorn Prelude, és a dir no hauria de caldre cap import. Si voleu usar una funció que requereixi import consulteu-ho abans amb el professor.

Excepte en els problemes 1.b i 4.a podeu usar les funcions auxiliars que us calguin.

Problema 1: *Suma sèrie*

Apartat 1.a: Definiu `genSum::[Integer]` com la llista infinta que conté en ordre creixent totes les sumes de la forma $\Sigma_{k=1}^{n} k$ per algun $n \geq 0$. És a dir,
`[0,1,3,6,10,15,....`

Apartat 1.b: Usant `genSum` (**obligatòriament**), feu una funció `esSum`, que donat un enter positiu, ens digui si és una suma de nombres consecutius començats per 1. Heu de fer una definició no recursiva que usi funcions predefinides de Haskell i `genSum`.

```
esSum n = ...
```

Problema 2: *Selecció de predicats*
Definiu una funció `selPred` que, donats un element x de tipus a i una llista lf de funcions de tipus $a \rightarrow Bool$, retorna un parell de llistes que contenen respectivament les funcions que avaluen a True i les que avaluen a False quan s'apliquen a x. Per exemple:
`selPred 8 [even,odd,(>3),(<=9),(==0)]` retorna `([even,(>3),(<=9)],[odd,(==0)])`

Com que les funcions no són de la classe `Show`, no es pot mostrar el resultat. Per això cal que definiu també la funció `check` que donat un element x de tipus a i una llista lf de funcions de tipus $a \rightarrow b$, retorna una llista de tipus $[b]$ amb els resultats d'aplicar les funcions a x. Per exemple:
`check 0 [odd,(==0)]` retorna `[False,True]`

Problema 3: *Arbres Fibonacci*
Cosidereu la següent definició del tipus abre binari

```
data Arbre a = Node a (Arbre a) (Arbre a)
             |Abuit
```

i la següent definició d'arbres de Fibonacci:

- L'arbre buit és un arbre de Fibonacci d'ordre 0.
- L'arbre que conté un únic node és un arbre de Fibonacci d'ordre 1.
- En altre cas, un árbre és de Fibonacci d'ordre n si el fill esquerre és un arbre de Fibonacci d'ordre n-1 i el fill dret és un arbre de Fibonacci d'ordre n-2.

Feu una funció `arbreFibonacci` que retorni -1 si l'arbre no és de Fibonacci i el seu ordre en cas contrari.

Problema 4: *Llistes Multiopció*
Volem representar el tipus (genèric) llista amb multiples opcions. És a dir, que cada posició de la llista no conté un únic element sinó que té varies opcions. Un element x pertany a una llista multiopció lm, si x és una de les opcions dels elements d'aluna posició de lm. Una llista normal l està *inclosa* en una llista multiopció lm, si l i lm tenen la mateixa logitud i cada element de l, està inclos entre les opcions de lm a la mateixa posició.

Apartat 4.a: Doneu la definició del tipus `MOList` i implementeu les operacions `inclosa` i `esta` (seguint les indicacions de l'enunciat).
Important: La funció `esta` l'heu de fer obligatòriament usant el `foldl` o el `foldr` (com a úniques funcions d'ordre superior) i operacions auxiliar de llistes predefinides.
La funció `inclosa` la podeu fer com preferiu.

Apartat 4.b: Definiu la igualtat de llistes multiopció, tenint en comte que dues llistes multiopció són iguals si tenen les mateixes opcions en cada posició. Indiqueu que `MOList` és una `instance` de la classe `Eq` on (==) és la igualtat que heu definit. Recordeu que cal que el tipus dels elements de la llista multiopció també siguin de la classe `Eq` (és important que no afegiu més condicions sobre aquests elements).