Parcial Llenguatges de Programació

Grau en Enginyeria Informàtica

9 Abril 2015

Per accedir al racó aneu a `https://examens.fib.upc.edu`
Cal que lliureu el codi via racó amb els comentaris que considereu necessaris en un arxiu "examen.hs" executable en l'entorn ghci sense activar cap opció addicional (només fent `ghci examen.hs`) i que solucioni els problemes que es llisten a continuació.
Imprimirem la vostra solució amb la comanda
`a2ps -1 -r -f 8 --borders=0 --no-header --header=Examen examen.hs -o examen.ps`
comproveu que el que envieu té una indentació correcte i no es surt dels límits de la pàgina.

Cal que al començar la solució de cada problema afegiu una línia comentada indicant el problema i subapartat que ve a continuació. Per exemple,
`-- Problema 3.1`
Es valorarà l'ús de funcions d'ordre superior predefinides. Ara bé, només es poden usar funcions predefinides de l'entorn Prelude: no podeu fer cap `import`.

Problema 1 (1 punt): *Concat.* Implementeu les funcions següents:
Apartat 1.1. Feu, **usant llistes per comprensió**, la funció `mconcat::[[a]]->[a]` que aplana una llista de llistes. Exemple:
`mconcat [[2,5],[],[8,8,2],[5]] = [2,5,8,8,2,5]`.
Apartat 1.2. Feu la funció `concat3::[[[a]]]->[a]` que aplana una llista de llistes de llistes (es valorarà la simplicitat de la solució). Exemple:
`concat3 [[[2,5]],[],[[],[8,8,2]],[[5]]] = [2,5,8,8,2,5]`.

Problema 2 (1 punt): *Fold2.* Feu la funció `fold2r::(a -> b -> c -> c) -> c -> [a] -> [b] -> c` que és com el `foldr` però rep dues llistes en lloc d'una i la funció paràmetre s'aplica com al `foldr` però agafant cada cop els dos primers elements de les llistes. És a dir, primer l'aplica als dos primers elements, després als dos segons, etc., fins que una de les dues llistes s'acaba. Per exemple,

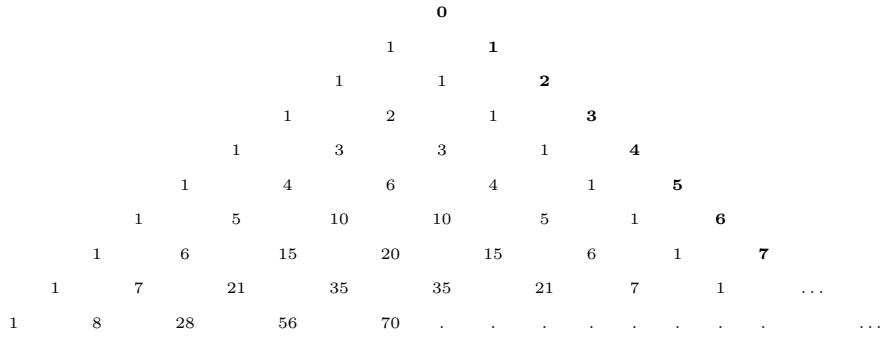
```
let ite x y z = if x then y:z else z
fold2r ite [] [True,False,False,True] [6,8,3,2,1,5,6]
```

és `[6,2]`.

Problema 3 (1 punt): *Mix.* Feu la funció `mix:: [a] -> [a] -> [a]` que ajunta les dues llistes agafant alternadament un element de cada llista, començant per la primera i si una de les dues acaba, posa tots els element de l'altre seguits. Per exemple, `mix [2,3,5,8] [6,2] = [2,6,3,2,5,8]`.

Feu la funció `lmix:: [Int] -> [a] -> [a]` que usa la llista d'enters per anar partint la segona llista i fer el `mix` de les dues parts (acumulativament). Per exemple, si la primera llista és `[5,3,8]` i la segona `"adfarbeco"`, primer fa el mix de `"adfar"` i `"beco"` que té com a resultat `"abdefcaor"`) i sobre aquesta continua fent els següents mix.
En total, `lmix [5,3,8] "adfarbeco"` és `"arebfdcao"`.

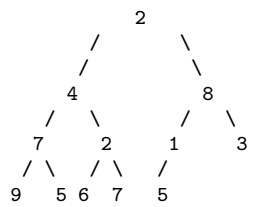
Problema 4 (2 punts): *Diagonals de Pascal.* Feu la funció `dPascal :: Int -> [Integer]` que retorna la llista infinita amb els valors del triangle de Pascal per la diagonal que indica el paràmetre. Tal com es mostra a la figura següent, entenem que la diagonal 0 és `[1,1,1,1,1,...]`, la diagonal 1 és `[1,2,3,4,5,...]`, la diagonal 2 és `[1,3,6,10,15,...]`, etc.



Exemple: `take 10 $ dPascal 5` és : `[1,6,21,56,126,252,462,792,1287,2002]`.

Problema 5 (2 punts): *Amplada.* Feu la definició del data `BTree` per arbres binaris polimòrfics amb els constructors `Node` i `Empty`. Feu que aquest data es pugui mostrar amb l'operació `show` de la forma més senzilla possible.

Feu la funció `buildTreeF :: [[a]] -> BTree a`, que rep una llista de llistes que representa els elements de cada nivell d'un arbre binari complet no necessàriament ple. Per exemple, per a la llista `[[2],[4,8],[7,2,1,3],[9,5,6,7,5]]`, l'arbre seria



Noteu que, per exemple, els heaps són arbres binaris complets no necessàriament plens. Podeu assumir que la llista de llistes és correcta, és a dir, representa un arbre binari.

Pista: Feu una funció auxiliar que retorni una llista d'arbres en lloc d'un sol arbre.

Exemple: `buildTreeF [[2],[4,8],[7,2,1]]` és

```
Node 2 (Node 4 (Node 7 Empty Empty) (Node 2 Empty Empty)) (Node 8 (Node 1 Empty Empty) Empty)
```

Problema 6 (3 punts): *Expressions.* Volem representar expressions generals sobre algun tipus i volem tenir una operació general d'avaluació. Per això feu els següents apartats:

Apartat 6.1. Definiu la classe `Lit`, on un tipus és d'aquesta classe si té les tres següents operacions: `unary`, que té un únic paràmetre i un resultat d'aquest tipus, `binary`, que té dos paràmetres i un resultat d'aquest tipus, i `list` que té un únic paràmetre que és una llista d'elements d'aquest tipus i el resultat és d'aquest tipus.

Apartat 6.2. Definiu un data polimòrfic `Expr a`, que representa expressions construïdes sobre valors d'un tipus genèric `a` amb els constructors `Val` que té un paràmetre de tipus `a`, `Unary` que té un paràmetre de tipus expressió, `Binary` que té dos paràmetres de tipus expressió i `List` que té un paràmetre que és una llista d'expressions. Feu que aquest data es pugui mostrar amb l'operació `show` de la forma més senzilla possible. Per exemple, podem definir

```
ex1 :: Expr Int
ex1 = Unary (Binary (List [Val 3, Unary (Val 2)]) (Val 8))
```

Apartat 6.3. Feu la funció `eval`, que rep una expressió genèrica `Expr a`, on `a` és de la classe `Lit`, i retorna un valor de tipus `a`. La funció `eval` ha d'avaluar l'expressió aplicant l'operació `unary` quan es troba el constructor `Unary`, l'operació `binary` quan es troba el constructor `Binary`, l'operació `list` quan es troba el constructor `List` i retornant el valor que conté quan es troba el constructor `Val`.

Apartat 6.4. Feu que `Int` sigui de la classe Lit, implementant `unary` com el canvi de signe, `binary` com la suma i `list` com la suma de tots els elements de la llista.

Així, finalment, si fem `eval ex1` el resultat és `-9`.