Haskell – Guia del laboratori 3

Estructures de dades

En Haskell també es poden crear tipus de dades personalitzats. Quan els determinem, cal triar el nom amb el que volem batejar la nova estructura, així com el nom dels mètodes constructors (els seus noms no necessàriament han de ser igual al nom del tipus, a diferència de Java).

Veiem els següents exemples:

Definició del tipus	Detalls
data Alumne = Alu String Double Double Double deriving (Eq,Show)	El nom del tipus és "Alumne" però el seu constructor és "Alu". Els atributs del tipus corresponen a: nom, nota1, nota2, nota3
data Cotxe = Cotxe String String Int deriving (Show)	Coincideix el nom del tipus amb el nom del constructor, ambdós son "Cotxe". Els atributs corresponen a: Marca, model i número de portes
data Point = Point Float Float deriving (Show)	Coincideix el nom del tipus amb el nom del constructor, ambdós son "Point". Els atributs corresponen a: coordenada x i coordenada y del punt.
data Figura = Circle Point Float | Rectangle Point Point deriving (Show)	El nom del tipus és "Figura" i té dos constructors diferents. El primer és "Circle" i seus atributs són el punt que determina el centre del cercle i el radi. El segon es "Rectangle" i els seus atributs són els punts de dues cantonades oposades.

Es fa servir "deriving (Eq,Show)" o variants d'igual manera que a Java utilitzavem "extends" o "implements" i capacitar al tipus d'operacions ja implementades.

Al definir la capçalera d'una funció que utilitza Datas, cal fer servir el nombre del tipus.

En canvi, quan implementem una funció que rep com a paràmetre un dels tipus, cal indicar-ho amb el nom del constructor i sempre entre parèntesi. A l'hora de fer la crida a la funció, els paràmetres també es passen utilitzant el nom del constructor.

A continuació veiem funcions que utilitzen els tipus anteriors:

Funció i crida	Descripció
surface :: Figura -> Float surface (Circle _ r) = pi * r ^ 2 surface (Rectangle (Point x1 y1) (Point x2 y2)) = (abs (x2 - x1)) * (abs (y2 - y1)) Exemple de crida: surface (Circle (Point 0 0) 1) Exemple de crida: surface (Rectangle (Point 0 0) (Point 2 2))	Funció que calcula la superfície de qualsevol figura
calificar :: [Alumne] -> [String] calificar [] = [] calificar ((Alu nom n1 n2 n3):rest) = (if (n1>= 4 && n2>=4 && n3>=4) then (if ((n1*2/10)+(n2*3/10)+(n3*5/10))>=5 then (nom++"->aprovat") else (nom++"->suspes")) else nom++"->suspes"):calificar rest Exemple de crida: calificar [(Alu "Juan" 6 5 4), (Alu "Pedro" 4 7 9)]	Funció que donada una llista d'alumnes calcula si aquests alumnes estan aprovats o suspesos. Cal treure un mínim de 4 a cada part per aprovar. La primera nota pondera un 20%, la segona un 30% i la tercera un 50%.

Àrbres

Els àrbres són simplement una estructura de dades recursiva, perquè cada branca d'un arbre és a la vegada un arbre en si mateixa.

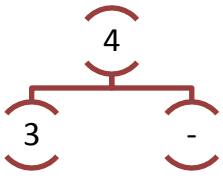
Definició	Descripció
data ArbreEnters = ArbreEnters ArbreEnters Int ArbreEnters | Null deriving (Show)	Els tres atributs que rep l'arbre corresponen a la branca esquerra, l'element central i la branca dreta. El constructor "Null" correspondria a un arbre sense cap element (i a la vegada a una branca inexistent).
Arbre d'elements genèrics: data Arbre a =Arbre (Arbre a) a (Arbre a) | Null' deriving (Show)	

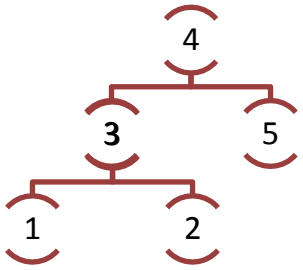
La forma d'instanciar-los és equivalent a com es fa a la resta de Datas:

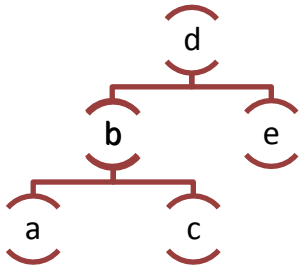
ArbreEnters (ArbreEnters Null 3 Null) 4 Null

Arbre (Arbre (Arbre Null' 1 Null') 3 (Arbre Null' 2 Null')) 4 (Arbre Null' 5 Null')

Arbre (Arbre (Arbre Null' "a" Null') "b" (Arbre Null' "c" Null')) "d" (Arbre Null' "e" Null')







La següent funció rep un paràmetre de tipus arbre i en retorna l'element màxim:

```
maxArbre :: ArbreEnters -> Int
maxArbre Null= -99999
maxArbre (ArbreEnters esquerra elem dreta) =
if ((maxArbre esquerra)>elem && (maxArbre esquerra)>(maxArbre dreta))
then maxArbre(esquerra)
else (if (elem>(maxArbre esquerra) && elem>(maxArbre dreta)) then elem
else (maxArbre dreta))
```

Exemples de crides a aquesta funció:

```
maxArbre (ArbreEnters (ArbreEnters Null 1 Null) 3 (ArbreEnters Null  5 Null))
maxArbre (ArbreEnters (ArbreEnters Null 1 Null) 7 (ArbreEnters Null  5 Null))
maxArbre (ArbreEnters (ArbreEnters Null 9 Null) 7 (ArbreEnters Null  5 Null))
```

Susana Bujalance García

Sèries infinites

Hem de conèixer dos funcions que ens vindran donades:

- La funció “numsfrom”: genera una sèrie infinita d’enters començant des del número enter que li passem com a paràmetre d’entrada
    - numsfrom 5 → 5,6,7,8...
- La funció “select”: passant-li una quantitat (n) i una llista infinita, retorna només els n primers de la llista infinita.
    - select 10 (numsfrom 10) → 10,11,12,13,14,15,16,17,18,19

La implementació de les funcions és la següent:

```
numsfrom :: Int -> [Int]
numsfrom n = n : numsfrom (n+1)

select :: Int -> [a] -> [a]
select n _ | n <= 0 = []
select _ [] = []
select n (x:xs) = x : select (n-1) xs
```

La combinació de les següents funcions permet generar la sèrie infinita de Fibonacci (0,1,1,2,3,5,8,13,... és a dir, cada nombre és la suma dels dos anteriors, sempre començant per 0 i 1):

```
--fibo rep un índex que referencia una posició de la sèrie i obté el Fibonacci que ocupa aquella posició
fibo :: Int -> Int
fibo 1 = 0
fibo 2 = 1
fibo n = fibo (n-1) + fibo (n-2)

fibonacci = map fibo (numsfrom 1)
```

Crida: select 15 fibonacci