Lenguajes de programación

Datos y Estructuras en Haskell

Tipos Predefinidos

- Atómicos y funcionales
  - Int, Integer, Char, Float, Double
  - 5 :: Int
  - ‘a’ :: Char
  - inc :: Int -> Int
- Estructurados
  - listas: [Integer], [Char], …
  - Tuplas: (‘b’,4) :: (Char,Integer), …
  - Combinaciones: ("Maria",[77,90,98])::(String,[Integer])

Elementos del mismo tipo

Elementos de tipos distintos

Ejemplo estructura de datos

- Se tiene una lista de registros de calificaciones de alumnos como

  [...[matrícula, nombre, [calificaciones]]...]

```
ejemplo = [[750706,"Ramiro Flores",
             [97,85,91]],
           [773454,"Myrna Vazquez",
             [98,75,88]],
           [764435,"Ruben Solis",
             [77,56,80]]]
```

Problema con los tipos estructurados

- ¿Cómo se interpretan?
  - `[[Integer]]`
  - `([Char],(Char,Char))`
  - `[[(Integer,Integer)]]`
- Error al intentar con la lista de sublistas:

```
ERROR - Cannot infer instance
*** Instance   : Num [Char]
*** Expression : [[750706,"Ramiro Flores",[97,85,91]],
                  [773454,"Myrna Vazquez",[98,75,88]],
                  [764435,"Ruben Solis",[77,56,80]]]
```

- Corregimos a:

```
ejemplo = [(750706,"Ramiro Flores",[97.0, 85.0, 91.0]),
           (773454,"Myrna Vazquez",[98.0, 75.0, 88.0]),
           (764435,"Ruben Solis",[77.0, 56.0, 80.0]) ]

Tipo ejemplo :: [(Integer,[Char],[Double])]
```

Problema ejemplo

- Escribe una función “promalum” que entrega el promedio de parciales de un alumno con registro:

```
[(Integer,[Char],[Double])]
```

- Haskell

```
promalum 773454 ejemplo => 87
```

Versión Haskell

```
promalum :: Integer ->
       [(Integer,[Char],[Double])]-> Double
promalum _ [] = 0.0
promalum mat1 ((mat2,_,parcialista):resto) =
    if mat1 == mat2 then
      sum parcialista /
        fromIntegral (length parcialista)
    else promalum mat1 resto
```

Representaciones "especiales"

- Las secuencias de números se pueden representar de varias maneras:

```
[1..4] => [1,2,3,4]
[1,4..12] => [1,4,7,10]
```

Salto = 4 – 1 = 3

Tipos definidos por el usuario

- Declaración por medio de la palabra reservada **data**
- Ejemplos:

```
data Boleano = Falso | Verdadero
    Falso :: Boleano
data Color = Rojo | Verde | Azul
    Rojo :: Color
data Punto a = Pt a a
    Pt 2.0 3.0 :: Punto Double
```

- Agregar **`deriving Show`** para que se pueda mostrar en la consola

Tipos recursivos

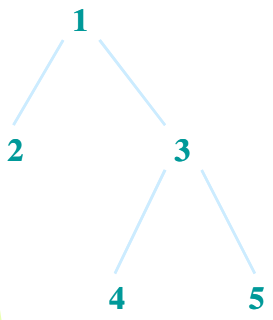
- No hay problema para definir tipos recursivos directamente
- Árboles Binarios

```
data Arbol e =
      Nodo (Arbol e) e (Arbol e)
      | ArbolVacio
```

- Listas Anidadas

```
data ListaAnidada a =
     Elemento a
     | Lista [ListaAnidada a]
```

Ejemplo árbol binario




```
ejemplo =
(Nodo
  (Nodo
    ArbolVacio
    2
    ArbolVacio)
  1
  (Nodo
    (Nodo
      ArbolVacio
      4
      ArbolVacio)
    3
    (Nodo
      ArbolVacio
      5
      ArbolVacio)
   )
)
```

Ejemplo

- **izq** arbol: entrega el subárbol izquierdo del árbol

```
izq ejemplo => Nodo ArbolVacio 2 ArbolVacio
```

```
izq :: Arbol a -> Arbol a
izq (Nodo l n r) = l
```

- **der** arbol: entrega el subárbol derecho del árbol

```
der ejemplo => Nodo (Nodo ArbolVacio 4
  ArbolVacio) 3 (Nodo ArbolVacio 5 ArbolVacio)))
```

```
der :: Arbol a -> Arbol a
der (Nodo l n r) = r
```

Ejemplo

- Cuenta los nodos de un árbol binario

```
numNodos ejemplo => 5
```

```
numNodos :: Arbol a -> Int
numNodos  ArbolVacio = 0
numNodos (Nodo l n r) =
   1 + (numNodos l) + (numNodos r)
```

Ejemplo

- Recorrido en entre-orden de un árbol binario

```
recorrido ejemplo => [2,1,4,3,5]
```

```
recorrido :: Arbol a -> [a]
recorrido ArbolVacio = []
recorrido (Nodo l n r) =
 (recorrido l)++[n]++(recorrido r)
```