Informática – Haskell – Matemáticas – Curso 2004-2005
Pepe Gallardo – Universidad de Málaga

Tema 1

1.1 Considérense las siguientes definiciones de funciones:

$$\begin{array}{lcl} inc & :: & Float \rightarrow Float \\ inc\ x & = & x + 1.0 \end{array}$$

$$\begin{array}{lcl} f & :: & Float \rightarrow Float \rightarrow Float \\ f\ x\ y & = & x + (4.0 * x) \end{array}$$

Sabiendo que si se intenta realizar una división por cero, la reducción de la expresión acaba con un error, reduce la expresión $f\ (inc\ 5.0)\ (7.0/0.0)$ utilizando orden aplicativo, orden normal y evaluación perezosa.

1.2 Sean las siguientes definiciones

$$\begin{array}{lcl} doble & :: & Integer \rightarrow Integer \\ doble\ x & = & x + x \end{array}$$

$$\begin{array}{lcl} cuadruple & :: & Integer \rightarrow Integer \\ cuadruple\ x & = & doble\ (doble\ x) \end{array}$$

Reduce la expresión $cuadruple\ (1 + 2)$ utilizando orden aplicativo, orden normal y evaluación perezosa.

Tema 2

2.1 En caso de que sean correctas, ¿Cuál es el tipo de las siguientes expresiones?. Si no son correctas indica porqué.

$(True,\ True)$
$(True,\ False)$
$(True,\ 'a',\ False)$
$(True,\ ('a',\ False))$
$isUpper$
$isUpper\ 'a'$
$(1 > 7,\ even\ 4,\ isUpper\ 'a')$
$['a',\ 'b',\ 'c']$
"abc"
$[True,\ 'a',\ False]$
$([True, False],\ 'a')$

2.2 Sabiendo que el máximo de dos números se puede calcular según la siguiente expresión

$$máximo(x, y) = \frac{(x + y) + |x - y|}{2}$$

- ✓ escribe una función $máximo$ en Haskell que tome dos enteros y devuelva el mayor.
- ✓ ¿Qué expresión escribirías en Haskell para calcular el máximo de 10 y 20 usando la función anterior?
- ✓ ¿Y el máximo de 10*3 y 40?

2.3 Usando la función que has definido en el apartado anterior, define una función en Haskell que devuelva el mayor de tres enteros. Escribe otra que devuelva el mayor de cuatro.

2.4 Escribe una función $entre0y9$ en Haskell que tome un entero y devuelva $True$ si está entre 0 y 9 o $False$ en caso contrario.

2.5 Escribe una función $esMúltiploDe3$ en Haskell que tome un entero y devuelva $True$ si éste es múltiplo de 3 o $False$ en caso contrario.

2.6 Escribe una función $coseno$ en Haskell que tome un ángulo en grados sexagesimales y devuelva su coseno.

Tema 3

3.1 ¿Cuáles de los siguientes patrones y argumentos unifican?. ¿Qué valores se asocian a cada variable en caso de que haya unificación?

Patrón	**Argumento**
0	0
x	0
$(x : ys)$	$[1, 2]$
$(x : ys)$	"sam"
$(x : ys)$	["sam"]
$(1 : xs)$	$[1, 2]$
$(1 : xs)$	$[2, 3]$
$(x : \_ : \_ : ys)$	$[1, 2, 3, 4, 5, 6]$
$[\,]$	$[\,]$
$[\ x\ ]$	["sam"]
$[1, x]$	$[1, 2]$
$[x, y]$	$[1]$
$x@\ y$	0
$a@(x : b@(y : zs))$	$[1, 2, 3, 4]$
$(n + 2)$	6
$(n + 1)$	0

3.2 Escribe una función

$$descomponer \ :: \ Integer \ \rightarrow \ (Integer, \ Integer, \ Integer)$$

que a partir de una cantidad de segundos, devuelva las horas, minutos y segundos equivalentes. Por ejemplo:

? $descomponer\ 7390$
$(2, 3, 10) \ :: \ (Integer, \ Integer, \ Integer)$

ya que 7390 segundos son dos horas, tres minutos y diez segundos. Da dos versiones, una usando **where** y otra usando **let in**.

3.3 Define una función $incTupla3$ que incremente todos los elementos de una tupla de tres enteros. Por ejemplo

? $incTupla3\ (1, 5, 8)$
$(2, 6, 9) \ :: \ (Integer, \ Integer, \ Integer)$

3.4 Define una función $incLista$ que incremente todos los elementos de una lista de enteros. Por ejemplo

? $incLista\ [1,5,8]$
$[2,6,9]\ \ ::\ [Integer]$

3.5 Escribe una función que determine si un año es bisiesto. Un año es bisiesto si es múltiplo de 4 (por ejemplo 1984). Una excepción a la regla anterior es que los años múltiplos de 100 sólo son bisiestos cuando a su vez son múltiplos de 400 (por ejemplo 1800 no es bisiesto, mientras que 2000 lo será):

? $esBisiesto\ 1984$
$True\ \ ::\ Bool$

3.6 Escribe una función recursiva que, usando sumas y restas (sin usar las funciones predefinidas div y mod), devuelva el resto de la división de dos enteros.

3.7 Escribe una función recursiva que, usando sumas y restas (sin usar las funciones predefinidas div y mod), devuelva el cociente que se obtiene al dividir dos números enteros.

3.8 Escribe una función recursiva $sumDesdeHasta$ que devuelva el sumatorio desde un valor entero hasta otro:

$$sumDesdeHasta\ a\ b \Longrightarrow \qquad a\ +\ (a+1)\ +\ (a+2)\ +\ \ldots\ +\ (b-1)\ +\ b$$

3.9 Escribe una función recursiva $prodDesdeHasta$ que devuelva el productorio desde un valor entero hasta otro:

$$prodDesdeHasta\ a\ b \Longrightarrow \qquad a\ *\ (a+1)\ *\ (a+2)\ *\ \ldots\ *\ (b-1)\ *\ b$$

3.10 Escribe una función $variaciones\ m\ n$ que calcule el número de variaciones de n elementos tomados de m en m. Usa para ello la siguiente relación:

$$variaciones\ m\ n = \frac{m!}{(m-n)!}$$

Escribe otra versión que use esta otra:

$$variaciones\ m\ n = (m-n+1) * (m-n+2) * \ldots (m-1) * m$$

3.11 Escribe una función recursiva que calcule números combinatorios usando la siguiente relación:

$$\left(\begin{array}{c} m \\ n \end{array} \right) = \frac{m!}{(m-n)! \cdot n!}$$

Escribe otra versión que use esta otra:

$$\begin{pmatrix} m \\ 0 \end{pmatrix} = 1$$
$$\begin{pmatrix} m \\ m \end{pmatrix} = 1$$
$$\begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} m-1 \\ n-1 \end{pmatrix} + \begin{pmatrix} m-1 \\ n \end{pmatrix}$$

¿Puedes garantizar que la última definición acaba?

3.12 Escribe una función que devuelva el i-ésimo número de la sucesión de *fibonacci*. Este sucesión tiene como primer término 0, como segundo 1, y cualquier otro término se obtiene sumado los dos que le preceden, es decir la secuencia es $0, 1, 1, 2, 3, 5, 8, 13, \ldots$. Por ejemplo:

? *fibonacci* 0
0 :: *Integer*

? *fibonacci* 6
8 :: *Integer*

El argumento de la función indica la posición dentro de la secuencia del elemento que queremos obtener. Consideraremos que el primer elemento de la secuencia tiene posición cero.

3.13 Escribe una función que determine el mayor de tres números enteros. Escribe otra para cuatro números.

3.14 Escribe una función *ordena*3 que tome tres números enteros y devuelva una terna con los números ordenados en orden creciente:

? *ordena*3 10 4 7
$(4, 7, 10)$ :: $(Integer,\ Integer,\ Integer)$

3.15 Escribe una función *esCapicúa* que determine si un número positivo de exactamente cuatro cifras es capicúa o no:

? *esCapicúa* 1221
True :: *Bool*

? *esCapicúa* 12
ERROR : *número de cifras incorrecto*

3.16 Escribe una función *sumaCifras* que calcule la suma de las cifras de un número natural, independientemente de su número de cifras:

? *sumaCifras* 123
6 :: *Integer*

? $sumaCifras\ 1235$
$11\ \ ::\ Integer$

3.17 Escribe una función que calcule el número de cifras de un número natural. Suponendremos que el número no tiene ceros a la izquierda:

? $números Cifras\ 123$
$3\ \ ::\ Integer$

3.18 Define una función

$$aEntero\ \ ::\ [Integer]\ \rightarrow\ Integer$$

que transforme una lista de dígitos en el correspondiente valor entero:

? $aEntero\ [2,\ 3,\ 4]$
$234\ \ ::\ Integer$

Define la función recíproca $aLista$:

? $aLista\ 234$
$[2, 3, 4]\ \ ::\ [Integer]$

Tema 4

4.1 Consideremos la siguiente función

$$\begin{array}{lcl} dosVeces & :: & (Integer\ \rightarrow\ Integer)\ \rightarrow\ Integer\ \rightarrow\ Integer \\ dosVeces\ f\ x & = & f\ (f\ x) \end{array}$$

✓ ¿Cuál es el tipo de la siguiente función?

$$fun\ \ =\ \ dosVeces\ (+1)$$

✓ Escribe una λ-expresión equivalente a la función fun.

✓ Escribe una sección equivalente a la función fun.

4.2 Define un función de orden superior recursiva $aplicaLista\ ::\ (Integer\ \rightarrow\ Integer)\ \rightarrow\ [Integer]\ \rightarrow\ [Integer]$ que aplique una función arbitraria (el primer parámetro) a todos los elementos de una lista, es decir

$$aplicaLista\ f\ [x_1,\ x_2,\ \ldots\ x_n] \Longrightarrow \qquad [f\ x_1,\ f\ x_2,\ \ldots\ f\ x_n]$$

Por ejemplo:

? $aplicaLista\ (+1)\ [1,2,3]$
$[2,3,4]\ ::\ [Integer]$

? $aplicaLista\ (*2)\ [1,2,3]$
$[2,4,6]\ ::\ [Integer]$

4.3 Define un función recursiva *potencia* que tome un exponente y base naturales y calcule la correspondiente potencia. Por ejemplo:

? $potencia\ 3\ 2$
$8\ ::\ Integer$

Escribe una expresión para elevar al cubo todos los elementos de la lista $[1,2,3,4,5]$ usando las funciones *aplicaLista* y *potencia*.

4.4 Escribe una función *derivada* que devuelva la derivada de una función de reales en reales usando la siguiente definición (Para aproximar el límite, simplemente evalúa la expresión con un valor pequeño para ϵ):

$$f'x = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - fx}{\epsilon}$$

Por ejemplo:

$coseno\ ::\ Float \to Float$
$coseno\ =\ derivada\ sin$

? $derivada\ sqrt\ 1.0$
$0.499487\ ::\ Float$

? $coseno\ 0.0$
$1.0\ ::\ Float$

¿Cuál es el tipo de la función *derivada*?

4.5 Escribe una función *logEnBase* que calcule el logaritmo de un número en una base dada usando la equivalencia:

$$\log_b x = \frac{\ln x}{\ln b}$$

Por ejemplo:

$log2\ ::\ Float \to Float$
$log2\ =\ logEnBase\ 2$

? $logEnBase\ 2\ 16$
$4.0\ ::\ Float$

? $log2\ 16$
$4.0\ ::\ Float$

¿Cuál es el tipo de la función $logEnBase$?

Tema 5

5.1 ¿Cuáles son los tipos polimórficos de las siguientes funciones?

$$
\begin{array}{ll}
swap\ (x, y) & = (y, x) \\
const\ x\ y & = x \\
subst\ f\ g\ x & = f\ x\ (g\ x) \\
pair\ (f, g)\ x & = (f\ x,\ g\ x) \\
cross\ (f, g)\ (x, y) & = (f\ x,\ g\ y)
\end{array}
$$

5.2 La función *flip* está predefinida del siguiente modo

$$
\begin{array}{ll}
flip & :: (a \to b \to c) \to b \to a \to c \\
flip\ f\ x\ y & = f\ y\ x
\end{array}
$$

Supongamos la siguiente definición de operador

$$(\gtrless) = flip\ (+\!\!+)$$

- ✓ ¿Cuál es el tipo de $(\gtrless)$?
- ✓ Calcula el resultado de la expresión $[1, 2, 3] \gtrless [5, 6]$
- ✓ A la vista del ejemplo, ¿para qué crees que sirve la función *flip*?

5.3 ¿Qué hace el siguiente operador?

$$
\begin{array}{l}
\textbf{infixr}\ 0\ >\$> \\
(>\$>) \quad :: (a \to a) \to a \to a \\
f\ >\$>\ x\ = f\ (f\ x)
\end{array}
$$

¿por qué su tipo no es $(>\$>) :: (a \to b) \to a \to b$?

5.4 Define el operador $(>\$>)$ del ejercicio anterior usando la composición de funciones $(.)$

5.5 Consideremos el siguiente operador:

$$
\begin{array}{l}
\textbf{infixl}\ 9\ >.> \\
f\ >.>\ g\ =\ \lambda x \to g\ (f\ x)
\end{array}
$$

- ✓ ¿Cuál es su tipo polimórfico?

✓ ¿Qué hace la siguiente función?

$$\begin{array}{l} fun \;::\; Integer \to Integer \\ fun \;=\; (+2) \;>.>\; (*2) \;>.>\; (+1) \end{array}$$

5.6 Consideremos la siguiente función polimórfica:

$$\begin{array}{l} iter \;::\; (Integer \to a \to a) \to a \to (Integer \to a) \\ iter\ f\ e \;=\; fun \\ \quad \textbf{where} \\ \quad\quad fun\ 0 \qquad\quad = e \\ \quad\quad fun\ m@(n+1) \;= f\ m\ (fun\ n) \end{array}$$

Cuya propiedad universal es:

$$g \;=\; iter\ f\ e \;\Leftrightarrow \left\{ \begin{array}{lcl} g\ 0 & = & e \\ g\ m@(n+1) & = & f\ m\ (g\ n) \end{array} \right\}$$

Define usando *iter* las siguientes funciones:

✓ *listaDecre* que dado un número natural x, construya una lista con la sucesión decreciente de naturales desde x a 1:

$$\begin{array}{l} \textbf{?}\ listaDecre\ 5 \\ [5,4,3,2,1] \;::\; [Integer] \end{array}$$

✓ *palos* que dado un número natural x, construya un cadena de caracteres con x letras I:

$$\begin{array}{l} \textbf{?}\ palos\ 3 \\ \text{"III"} \;::\; String \end{array}$$

5.7 La función predefinida *reverse* invierte el orden de los elementos de una lista. Por ejemplo

$$\begin{array}{l} \textbf{?}\ reverse\ [1,2,3] \\ [3,2,1] \;::\; [Integer] \end{array}$$

Defínela y da su tipo polimórfico. ¿Cuál es es tipo de la función $palíndromo\ xs \;=\; (reverse\ xs == xs)$?

5.8 ¿Cuál es el tipo polimórfico de la función *twice*?

$$twice\ f\ x \;=\; f\ (f\ x)$$

¿Cuál es el valor de cada una de las siguientes expresiones?

$twice\ (+1)\ 0$
$twice\ twice\ (+1)\ 0$

5.9 La función predefinida *zip* empareja dos listas hasta que una de ellas se agote. Por ejemplo

? $zip\ [1,2,3]\ [4,5,6]$
$[(1,4),\ (2,5),\ (3,6)]\ \ ::\ [(Integer, Integer)]$

? $zip\ [1,2,3]\ [True,\ False]$
$[(1, True),\ (2, False)]\ \ ::\ [(Integer, Bool)]$

? $zip\ [True,\ False]\ [1,2,3]$
$[(True, 1),\ (False, 2)]\ \ ::\ [(Bool, Integer)]$

Defínela y da su tipo polimórfico.

5.10 ¿Cuál es el tipo de las siguientes expresiones (en caso de que sean correctas)?

- ✓ $not\ .\ even$
- ✓ $even\ .\ not$
- ✓ $chr\ .\ ord$
- ✓ $ord\ .\ chr$
- ✓ $ord\ .\ chr\ .\ (+1)$
- ✓ $map\ not$
- ✓ $map\ (\lambda\, x\ \rightarrow\ not\ x)$
- ✓ $map\ (not\ .\ even)$
- ✓ $map\ not\ [True, False]$
- ✓ $map\ ord$
- ✓ $map\ 1$
- ✓ $map\ (+1)$
- ✓ $map\ (map\ (+1))$
- ✓ $map\ (+\!\!+[1])$
- ✓ $map\ (1\ :)$

¿Cuál es el valor de la expresión $map\ (map\ (+1))\ [[1,2,3],\ [10,11]]$?

Tema 6

6.1 Dado el siguiente tipo para representar temperaturas expresadas en dos escalas:

$$\textbf{data}\ \mathit{Temp}\ =\ \mathit{Centígrado}\ \mathit{Float} \mid \mathit{Fahrenheit}\ \mathit{Float}\ \textbf{deriving}\ \mathit{Show}$$

de modo que, por ejemplo, $\mathit{Centígrado}\ 10$ representa 10 grados centígrados y $\mathit{Fahrenheit}\ 15$ representa 15 grados Fahrenheit, escribe una función $\mathit{estáCongelada}\ ::\ \mathit{Temp} \to \mathit{Bool}$ que compruebe si a la temperatura que toma como parámetro el agua está congelada o no. Recuerda que el agua se congela a cero grados centígrados o a 32 grados Fahrenheit.

Para pasar de grados Centígrado a grados Fahrenheit, se usa la siguiente conversión:

$$^{\circ}F = 9/5 \cdot {}^{\circ}C + 32^{\circ}$$

Para pasar de grados Fahrenheit a grados Centígrado, se usa la siguiente conversión:

$$^{\circ}C = ({}^{\circ}F - 32^{\circ}) \cdot 5/9$$

Escribe dos funciones *aCentígrado* y *aFahrenheit* que permitan transformar temperaturas entre las dos escalas:

```
? aFahrenheit (Centígrado 0)
Fahrenheit 32  :: Temp
```

6.2 Define una función que calcule el perímetro de una *Figura* según están definidas en las transparencias del tema.

6.3 Dadas las siguientes declaraciones de tipo:

```
type Real      = Float
type Imag      = Float
data Complejo  = Real :- Imag  deriving Show
data Resultado = UnaReal Float
               | DosReales Float Float
               | DosComplejas Complejo Complejo
                 deriving Show
```

completa la función

```
raíces       :: Float → Float → Float → Resultado
raíces a b c = ...
```

que devuelva las raíces de la ecuación de segundo grado $ax^2 + bx + c = 0$. dados sus coeficientes. Si la ecuación tiene una solución real se usará el primer

constructor para la solución, si tiene dos soluciones reales se usará el segundo y si tiene dos soluciones complejas se usará el tercero.

6.4 Para el tipo Nat definido en las transparencias del tema, define

- ✓ Un operador para restar dos naturales
- ✓ Un operador para calcular la potencia de naturales
- ✓ Una función que convierta un valor de tipo $Integer$ en el correspondiente de tipo Nat
- ✓ Una función que convierta un valor de tipo Nat en el correspondiente de tipo $Integer$
- ✓ Dos funciones $divNat$ y $modNat$ que calculen el cociente y resto de dividir dos naturales

6.5 Define las siguientes funciones como concreciones de $foldNat$:

$$
\begin{array}{lcl}
suma & :: & Nat \rightarrow Nat \rightarrow Nat \\
suma\ m\ Cero & = & m \\
suma\ m\ (Suc\ n) & = & Suc\ (suma\ m\ n)
\end{array}
$$

$$
\begin{array}{lcl}
prod & :: & Nat \rightarrow Nat \rightarrow Nat \\
prod\ m\ Cero & = & Cero \\
prod\ m\ (Suc\ n) & = & suma\ m\ (prod\ m\ n)
\end{array}
$$

Ayuda: Observa que la primera definición, por ejemplo, es equivalente a:

$$
\begin{array}{lcl}
suma & :: & Nat \rightarrow Nat \rightarrow Nat \\
(suma\ m)\ Cero & = & m \\
(suma\ m)\ (Suc\ n) & = & Suc\ ((suma\ m)\ n)
\end{array}
$$

6.6 El siguiente tipo puede ser utilizado para representar expresiones aritméticas simples sobre enteros:

$$
\begin{array}{lcl}
\textbf{data}\ Expr & = & Valor\ Integer \\
& | & Expr\ :+:\ Expr \\
& | & Expr\ :-:\ Expr \\
& | & Expr\ :*:\ Expr \\
& & \textbf{deriving}\ Show
\end{array}
$$

Por ejemplo

$$
\begin{array}{lcl}
ej1 & :: & Expr \\
ej1 & = & Valor\ 5 \quad \textit{-- representa el valor 5}
\end{array}
$$

$$
\begin{array}{lcl}
ej2 & :: & Expr \\
ej2 & = & ej1\ :+:\ Valor\ 3 \quad \textit{-- representa la expresión 5 + 3}
\end{array}
$$

```
ej3 :: Expr
ej3 = ej2 :*: Valor 10  -- representa la expresión (5 + 3) * 10
```

Escribe funciones que calculen:

- ✓ El número de operadores que aparece en una expresión
- ✓ El valor de una expresión
- ✓ El número de constantes que aparece en una expresión

```
? numOperadores ej3
2 :: Integer
? valor ej3
80 :: Integer
? numConstantes ej3
3 :: Integer
```

6.7 Sea el siguiente tipo para representar listas:

```
infixl 5 :<
data SnocList a = Vacía | (SnocList a) :< a deriving Show
```

de modo que una lista de la forma $xs :< x$ representa una lista cuyo último elemento es x y el segmento inicial es xs. Por ejemplo:

```
l :: SnocList Integer
l = Vacía :< 1 :< 2 :< 3
```

es una lista cuyo primer elemento es 1, el segundo es 2 y el tercero 3. Para este tipo, define:

- ✓ una función que devuelva la cabeza (primer elemento) de una lista
- ✓ una función que devuelva el último elemento de una lista
- ✓ una función que calcule la longitud de una lista
- ✓ un operador $++$ para concatenar dos listas
- ✓ una función *mapSnocList* análoga a *map* para este tipo de listas

6.8 Sea el siguiente tipo para representar listas:

```
infixl 5 :+:
data ConcatList a = V | U a | ConcatList a :+: ConcatList a deriving Show
```

donde V representa la lista vacía, $U\ x$ permite representar listas con un único elemento y $:+:$ permite formar una lista concatenando otras dos. Por ejemplo:

$$
\begin{array}{lcl}
l & :: & ConcatList\ Integer \\
l & = & U\ 1\ :+:\ U\ 2
\end{array}
$$

Para este tipo, define:

- ✓ una función que devuelva la cabeza (primer elemento) de una lista
- ✓ una función que devuelva el último elemento de una lista
- ✓ una función que calcule la longitud de una lista
- ✓ un operador $++$ para concatenar dos listas
- ✓ una función $mapConcatList$ análoga a map para este tipo de listas

Tema 7

7.1 ¿Son correctas las siguientes definiciones de función? Si son correctas, ¿cuál es el tipo de f en cada caso?

1. $f\ x\ y\ =\ \textbf{if}\ (x == y)\ \textbf{then}\ 5\ \textbf{else}\ True$

2. $f\ x\ y\ =\ \textbf{if}\ (x\ \&\&\ y)\ \textbf{then}\ 4\ \textbf{else}\ 1.5$

3.
$$
\begin{array}{lcl}
f\ [\,]\ \_ & = & 0 \\
f\ (x : xs)\ True & = & length\ (x : xs) \\
f\ (x : xs)\ False & = & negate\ (length\ (x : xs))
\end{array}
$$

7.2 Sea el siguiente tipo para representar números racionales:

$$
\begin{array}{l}
\textbf{infix}\ 9\ :/ \\
\textbf{data}\ Racional\ =\ Integer\ :/\ Integer
\end{array}
$$

donde el primer entero es el numerador y el segundo es el denominador. Haz este tipo instancia de las clases Eq, Ord, $Show$, Num y $Fractional$.

7.3 Sea el siguiente tipo para representar números naturales:

$$\textbf{data}\ Nat\ =\ Cero \mid Suc\ Nat\ \textbf{deriving}\ Show$$

Haz este tipo instancia de las clases Eq, Ord, y Num.

7.4 La clase predefinida $Functor$

```
class Functor f where
  fmap :: (a → b) → f a → f b
```

permite generalizar la función de listas *map* a estructuras polimórficas arbitrarias. Por ejemplo, las instancias predefinidas de esta clase para listas y el tipo *Maybe* son:

```
instance Functor [] where
  fmap = map

instance Functor Maybe where
  fmap f Nothing  = Nothing
  fmap f (Just x) = Just (f x)
```

```
? fmap (+1) [1,2,3]
[4,5,6] :: [Integer]

? fmap (+1) (Just 2)
Just 3 :: Maybe Integer
```

Realiza instancias para esta clase con los tipos *SnocList* y *ConcatList* definidos en los ejercicios del tema previo.

7.5 Define la función *ocurrencias* que toma una lista y un dato y devuelve el número de veces que aparece el dato en la lista:

```
? ocurrencias 2 [1,2,3,2,4,5,2]
3 :: Integer

? ocurrencias 'a' "la casa roja"
4 :: Integer
```

¿Cuál es el tipo de esta función? ¿Cómo puedes utilizar esta función para definir la función *pertenece* que compruebe si un elemento aparece en una lista?

```
? pertenece 2 [1,2,3,2,4,5,2]
True :: Bool

? pertenece 8 [1,2,3,2,4,5,2]
False :: Bool
```

7.6 Define una instancia de *Eq* y otra de *Ord* par el tipo *Color*

```
data Color = Violeta | Azul | Verde | Amarillo | Naranja | Rojo
             deriving Show
```

teniendo en cuenta que dos colores solo son iguales si son idénticos y que el orden de los colores viene dado por la enumeración anterior (no derives las instancias, defínelas).

7.7 Considérese las siguientes definiciones de tipos:

$$
\begin{array}{lcl}
\textbf{type}\ \mathit{Lado} & = & \mathit{Float} \\
\textbf{type}\ \mathit{Radio} & = & \mathit{Float} \\
\textbf{data}\ \mathit{Cuadrado} & = & \mathit{UnCuadrado\ Lado}\ \textbf{deriving}\ \mathit{Show} \\
\textbf{data}\ \mathit{Rectángulo} & = & \mathit{UnRectángulo\ Lado\ Lado}\ \textbf{deriving}\ \mathit{Show} \\
\textbf{data}\ \mathit{Círculo} & = & \mathit{UnCírculo\ Radio}\ \textbf{deriving}\ \mathit{Show} \\
\textbf{data}\ \mathit{Cubo} & = & \mathit{UnCubo\ Lado}\ \textbf{deriving}\ \mathit{Show} \\
\textbf{data}\ \mathit{Esfera} & = & \mathit{UnaEsfera\ Radio}\ \textbf{deriving}\ \mathit{Show}
\end{array}
$$

Define una clase *TieneÁrea* que incluya un método para calcular el área de una figura. Realiza las instancias correspondientes para los tipos anteriores, de modo que sea posible calcular el área para cada tipo de figura. Define también una clase *TieneVolumen* para calcular el volumen de figuras. Realiza las instancias correspondientes:

? *área* (*UnCuadrado* 2.0)
4.0 :: *Float*

? *área* (*UnRectángulo* 2.0 4.0)
8.0 :: *Float*

? *volumen* (*UnCubo* 3.0)
27.0 :: *Float*

Tema 8

8.1 Define

✓ una función *partir* que parta una lista en dos mitades.

? *partir* $[7, 3, 8, 4]$
$([7, 3], [8, 4])$ :: $([Integer], [Integer])$

✓ una función *mezclar* que tome dos listas ordenadas y devuelve una nueva lista con la mezcla ordenada de ambas.

? *mezclar* $[3, 7]$ $[4, 8]$
$[3, 4, 7, 8]$ :: $[Integer]$

✓ usando las funciones anteriores, define la función *mergeSort* que ordene una lista por el método de ordenación por mezcla. Este método consiste en ordenar una lista partiéndola en dos mitades, ordenando cada mitad y mezclando por último las mitades ordenadas.

? $mergeSort\ [7,2,8,4]$
$[2,4,7,8]\ \ ::\ [Integer]$

8.2 Define una función $estáOrdenada$ que compruebe si una lista está ordenada ascendentemente.

? $estáOrdenada\ [3,7,9]$
$True\ \ ::\ Bool$

? $estáOrdenada\ [3,8,7,9]$
$False\ \ ::\ Bool$

8.3 Define una función que inserte un dato en su posición adecuada dentro de una lista ordenada.

? $insertar\ 5\ [1,3,9]$
$[1,3,5,9]\ \ ::\ Bool$

8.4 Define

- ✓ una función que devuelva todos los segmentos iniciales de una lista:

  ? $inits\ [1,2,3]$
  $[\,[\,],\ [1],\ [1,2],\ [1,2,3]\,]\ \ ::\ [[Integer]]$

- ✓ una función que devuelva los segmentos finales de una lista:

  ? $tails\ [1,2,3]$
  $[\,[1,2,3],\ [2,3],\ [3],\ [\,]\,]\ \ ::\ [[Integer]]$

- ✓ una función que devuelva los segmentos de datos consecutivos de una lista:

  ? $segs\ [1,2,3]$
  $[\,[\,],\ [1],\ [2],\ [3],\ [1,2],\ [2,3],\ [1,2,3]\,]\ \ ::\ [[Integer]]$

- ✓ una función que devuelva las partes (todas las sublistas) de una lista:

  ? $partes\ [1,2,3]$
  $[\,[\,],\ [1],\ [2],\ [3],\ [1,2],\ [2,3],\ [1,3],\ [1,2,3]\,]\ \ ::\ [[Integer]]$

Nota: no es necesario que los resultados aparezcan en el mismo orden que en los ejemplos.

8.5 Da la definición de las funciones predefinidas *head*, *tail*, *last*, *init*, *take*, *drop*, (!!), *takeWhile*, *sum*, *product*, *maximum*, *minimum*, *zip*, *unzip* y *zipWith*.

8.6 Expresa como concreciones de *foldr* y *foldl* las funciones predefinidas *length*, *map f*, *filter p*, *concat*, y *unzip*.

8.7 Consideremos las siguientes funciones:

- ✓ *nub* que elimina elementos duplicados de una lista:

  ? $nub\ [1,2,3,2,4,2,1]$
  $[1,2,3,4] \quad :: \ [Integer]$

- ✓ *intersperse* que intercala un elemento entre dos consecutivos de una lista:

  ? $intersperse\ 0\ [1,2,3]$
  $[1,0,2,0,3] \quad :: \ [Integer]$

- ✓ *isPreffixOf* que comprueba si una lista es prefijo de otra:

  ? "ab" ‘*isPreffixOf*‘ "abcd"
  $True \quad :: \ Bool$

- ✓ *isSuffixOf* que comprueba si una lista es sufijo de otra
- ✓ *delete* que elimina la primera ocurrencia de un dato de una lista:

  ? $delete\ 1\ [2,1,3,1,1]$
  $[2,3,1,1] \quad :: \ [Integer]$

- ✓ (\\) que realiza la diferencia de listas:

  ? $[1,1,2,3,1]$ \\ $[1,1,3]$
  $[2,1] \quad :: \ [Integer]$

Da sus definiciones.

8.8 Usando listas por comprensión, define:

- ✓ una función $expandir \quad :: \ [Integer] \rightarrow [Integer]$ que reemplace en una lista de números positivos cada número n por n copias de sí mismo:

  ? $expandir\ [1..4]$
  $[\,1,2,2,3,3,3,4,4,4,4] \quad :: \ [Integer]$

✓ la función *concat*

8.9 Mientras que las funciones de plegado pueden ser utilizadas para expresar funciones que consumen listas, las funciones de desplegado se usan para expresar funciones que producen listas. Consideremos las siguiente función de desplegado

```
unfold           :: (b → Bool) → (b → a) → (b → b) → b → [a]
unfold p h t x   = fun x
  where
    fun x
      | p x        = []
      | otherwise  = h x : fun (t x)
```

que encapsula un patrón de cómputo para producir una lista a partir de cierta semilla x. Si la condición p es cierta para la semilla, entonces se produce la lista vacía (se deja de generar la lista resultado). En otro caso, el resultado es una lista cuya cabeza se obtiene aplicando h a la semilla y cuya cola es una llamada recursiva con semilla $t\ x$. Por ejemplo, la función $desdeHasta\ n\ m$ que genera la lista $[n..m]$

```
desdeHasta       :: Integer → Integer → [Integer]
desdeHasta n m   = fun n
  where
    fun n
      | n > m      = []
      | otherwise  = n : fun (n + 1)
```

puede ser definida como concreción de *unfold*:

```
desdeHasta       :: Integer → Integer → [Integer]
desdeHasta n m   = unfold (> m) id (+1) n
```

✓ evalúa paso a paso la expresión $desdeHasta\ 1\ 3$ usando esta última definición

✓ define como concreción de *unfold* la función $cuadradosDesdeHasta\ n\ m$ que devuelva $[x^{\wedge}\ 2 \mid x \leftarrow [n..m]]$

✓ define como concreción de *unfold* la función $desde\ n$ que devuelva la lista $[n..]$

✓ define como concreción de *unfold* la función $identidad :: [a] \rightarrow [a]$ que devuelva su argumento inalterado

✓ define como concreción de *unfold* la función $map\ f$

8.10 Se pretende escribir una función *pascal* que devuelva una lista infinita de listas, donde cada lista es una fila del triángulo de *Pascal*:

```
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 ...........
```

A partir de las siguientes observaciones,

```
  0 1       0 1 1      0 1 2 1
+ 1 0     + 1 1 0    + 1 2 1 0
-----     -------    ---------    ...
  1 1       1 2 1      1 3 3 1
```

define la función *pascal* en Haskell. Por ejemplo:

? $take\ 5\ pascal$
$[[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]] \ :: \ [[Integer]]$

8.11 Construye funciones que devuelvan listas infinitas con los términos de las siguientes series:

$$expo(x) = 1 + x + \tfrac{1}{2!}x^2 + \tfrac{1}{3!}x^3 + \ldots$$

$$seno(x) = x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \ldots$$

$$coseno(x) = 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \ldots$$

Usa las series para calcular aproximaciones a e, $seno(0)$ y $coseno(0)$.

Tema 9

9.1 Para el tipo $\acute{A}rbolB$ definido en las transparencias, define

- ✓ una función $map\acute{A}rbolB \ :: \ (a \to b) \to \acute{A}rbolB\ a \to \acute{A}rbolB\ b$, similar a map para listas, que aplique una función a todos los nodos de un árbol
- ✓ define esta función también para árboles generales.

9.2 Define los recorridos en pre-orden y post-orden para árboles generales.

9.3 Para el tipo $\acute{A}rbolB$ definido en las transparencias, define

- ✓ una función $todosÁrbolB :: (a \to Bool) \to ÁrbolB\ a \to Bool$ que compruebe si todos los datos almacenados en un árbol cumplen una condición
- ✓ una función $algunoÁrbolB :: (a \to Bool) \to ÁrbolB\ a \to Bool$ que compruebe si alguno de los datos almacenados en un árbol cumple una condición
- ✓ define estas funciones también para árboles generales.

9.4 Consideremos el siguiente tipo para representar árboles binarios con datos de tipo a almacenados tan solo en sus hojas

$$\textbf{data}\ ÁrbolH\ a = HojaH\ a \mid NodoH\ (ÁrbolH\ a)\ (ÁrbolH\ a)\ \textbf{deriving}\ Show$$

- ✓ define las funciones $tamañoH$, $profundidadH$, $todosÁrbolH$ y $algunoÁrbolH$ para este tipo de árbol.

9.5

- ✓ Define una función $máximoB$, similar a $maximum$ para listas, que devuelva el máximo valor de un árbol binario. Escribe también el tipo de esta función.
- ✓ define esta función también para árboles generales.

9.6

- ✓ Define una función $ocurrenciasB$ que devuelva cuántas veces aparece un dato en un árbol binario no necesariamente de búsqueda.
- ✓ define esta función también para árboles generales.

9.7 Define una función $hojas$ que devuelva los nodos hoja de un árbol (aquellos que no tienen hijos). Da versiones para árboles binarios y generales.

Tema 10

10.1 Para la siguiente definición del operador $(<+>)$,

$$\begin{array}{lcl} (<+>) & :: & Nat \to Nat \to Nat \\ m\ <+>\ Cero & = & m \\ m\ <+>\ Suc\ n & = & Suc\ (m\ <+>\ n) \end{array}$$

demostrar que cumple la propiedad asociativa

$$\forall\ x,\ y,\ z :: Nat\ \textbf{.}\ (x\ <+>\ y)\ <+>\ z\ \equiv\ x\ <+>\ (y\ <+>\ z)$$

10.2 Para el operador $(<+>)$ del ejercicio anterior

- ✓ demuestra que cumple los siguientes lemas

$$\forall\ x :: Nat\ \textbf{.}\ Cero\ <+>\ x\ \equiv\ x$$
$$\forall\ x,\ y :: Nat\ \textbf{.}\ Suc\ x\ <+>\ y\ \equiv\ Suc\ (x\ <+>\ y)$$

- ✓ utilizando los lemas, demuestra que el operador $(<+>)$ es conmutativo

$$\forall\ x,\ y :: Nat\ \textbf{.}\ x\ <+>\ y\ \equiv\ y\ <+>\ x$$

10.3 Demuestra que map distribuye sobre la composición $(.)$, usando las definiciones estándar de estas funciones

$$map\ (f.g)\ \equiv\ map\ f\ .\ map\ g$$

es decir,

$$\forall\ xs :: [a],\ g :: a\ \to\ b,\ f :: b\ \to\ c\ \textbf{.}$$
$$map\ (f\ .\ g)\ xs\ \equiv\ (map\ f\ .\ map\ g)\ xs$$

10.4 Para la definición estándar de $(+\!\!+)$ demuestra que se cumplen las siguientes propiedades

- ✓ $(+\!\!+)$ es asociativo

$$\forall\ xs,\ ys,\ zs :: [a]\ \textbf{.}\ (xs\ +\!\!+\ yz)\ +\!\!+\ zs\ \equiv\ xs\ +\!\!+\ (yz\ +\!\!+\ zs)$$

- ✓ $[\,]$ es elemento neutro por la derecha

$$\forall\ xs :: [a]\ \textbf{.}\ xs\ +\!\!+\ [\,]\ \equiv\ xs$$

- ✓ $[\,]$ es elemento neutro por la izquierda

$$\forall\ xs :: [a]\ \textbf{.}\ [\,]\ +\!\!+\ xs\ \equiv\ xs$$

10.5 Demuestra que las siguientes funciones son equivalentes (calculan lo mismo),

$$\begin{array}{lcl} suma & :: & [Int]\ \to\ Int \\ suma\ [\,] & = & 0 \\ suma\ (x : xs) & = & x\ +\ suma\ xs \end{array}$$

$$\begin{array}{l} suma' \ :: \ [Int] \ \to \ Int \\ suma' \ = \ foldr \ (+) \ 0 \end{array}$$

es decir, demuestra

$$\forall \ xs :: [Int] \ \textbf{.} \ suma \ xs \ \equiv \ suma' \ xs$$

Nota: Utiliza la definición estándar de *foldr*:

$$\begin{array}{l} foldr \ f \ e \ [\,] \ = \ e \\ foldr \ f \ e \ (x : xs) \ = \ f \ x \ (foldr \ f \ e \ xs) \end{array}$$

10.6 Demuestra que *map* no modifica la longitud de una lista:

$$\forall \ xs :: [a], \ f :: a \ \to \ b \ \textbf{.} \ length \ (map \ f \ xs) \ \equiv \ length \ xs$$

10.7 Demuestra que las dos funciones

$$\begin{array}{l} todosÁrbolB \ :: \ (a \ \to \ Bool) \ \to \ ÁrbolB \ a \ \to \ Bool \\ todosÁrbolB \ p \ VacíoB \ = \ True \\ todosÁrbolB \ p \ (NodoB \ i \ r \ d) \ = \ p \ r \ \&\& \\ \qquad\qquad\qquad\qquad\quad todosÁrbolB \ p \ i \ \&\& \ todosÁrbolB \ p \ d \end{array}$$

$$\begin{array}{l} todosÁrbolB' \ :: \ (a \ \to \ Bool) \ \to \ ÁrbolB \ a \ \to \ Bool \\ todosÁrbolB' \ p \ = \ foldÁrbolB \ (\lambda \ ti \ r \ td \ \to \ p \ r \ \&\& \ ti \ \&\& \ td) \ True \end{array}$$

son equivalentes:

$$\forall \ t :: ÁrbolB \ a, \ \forall \ p :: a \ \to \ Bool \ \textbf{.} \ todosÁrbol \ p \ t \ \equiv \ todosÁrbol' \ p \ t$$

Nota: Utiliza la definición de *foldÁrbolB* de las transparencias:

$$\begin{array}{l} foldÁrbolB \ f \ z \ VacíoB \ = \ z \\ foldÁrbolB \ f \ z \ (NodoB \ i \ r \ d) \ = \ f \ (foldÁrbolB \ f \ z \ i) \ r \ (foldÁrbolB \ f \ z \ d) \end{array}$$

10.8 Enuncia el principio de inducción para el tipo *ÁrbolH*

$$\textbf{data} \ ÁrbolH \ a \ = \ HojaH \ a \mid NodoH \ (ÁrbolH \ a) \ (ÁrbolH \ a)$$

Demuestra, mediante dicho principio, que la función

$$\begin{array}{l} espejoH \ :: \ ÁrbolH \ a \ \to \ ÁrbolH \ a \\ espejoH \ (HojaH \ x) \ = \ (HojaH \ x) \\ espejoH \ (NodoH \ i \ d) \ = \ NodoH \ (espejoH \ d) \ (espejoH \ i) \end{array}$$

verifica la siguiente propiedad:

$$\forall\ t :: \mathit{ÁrbolH}\ a\ \textbf{.}\ \mathit{espejoH}\ (\mathit{espejoH}\ t) \equiv t$$