# **INFORMATICA**.
EXAMEN PARCIAL Febrero 2003. Haskell

**Apellidos, Nombre** | **Grupo**

**1)** La función `and` toma una lista de booléanos y devuelve `True` si todos los elementos de la lista son `True`. Por ejemplo:

```
and [1 < 2, True, even 4]  ==> True
and [1 < 2, False, even 4] ==> False
```

a) (*1 pto*) Define la función `and` de modo recursivo, indicando además su tipo.
b) (*1 pto*) Define la función `and` usando `foldr`.

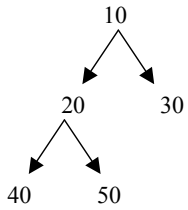
**2)** La función de orden superior `all` toma una función y una lista de valores y devuelve `True` si el resultado de aplicar la función argumento a cada elemento de la lista es `True`. Por ejemplo:

```
all even [10, 12, 10+4]        ==> True
all (>10.5) [20.3, 12.0, 1.4]  ==> False
```

a) (*1 pto*) Define la función `all` de modo recursivo, indicando además su tipo polimórfico.
b) (*1 pto*) Define la función `all` usando `foldr`.
c) (*1 pto*) Define la función `all` usando `and` y `map`.

**3)** a) (*1.5 ptos*) Define una función `hastaNivelB`, que tome un número y un árbol binario y devuelva una lista con todos los nodos del árbol cuyo nivel no supere el indicado. Considérese que el dato en la raíz del árbol está a nivel 0 y que el nivel aumenta en uno cada vez que se desciende en el árbol.

Por ejemplo, en el siguiente árbol




el dato 10 está a nivel 0, los datos 20 y 30 a nivel 1 y los datos 40 y 50 a nivel 2. Si la variable `a` representa el árbol anterior, algunos ejemplos son

```
hastaNivelB 0 a ==> [10]
hastaNivelB 1 a ==> [10,20,30]
hastaNivelB 2 a ==> [10,20,30,40,50]
```

b) (*1.5 ptos*) Define la función `hastaNivel` similar a la anterior pero para árboles generales.

# **INFORMATICA**.

EXAMEN PARCIAL Febrero 2003. Haskell

**Apellidos, Nombre** | **Grupo**

**Test** (*2 puntos*)

Para cada pregunta hay una única respuesta correcta. Las respuestas correctas puntúan +0.25 puntos y las incorrectas –0.08 puntos . Las respuestas en blanco no puntúan. Solo se corregirán los resultados escritos en la tabla que aparece al final.

**1**. ¿Cuál de las siguientes expresiones no es una lista válida en Haskell?

*a)* *[ 1 ,2 ,3 ]*
*b)* *[ (1 ,2) , (3 ,4, 5) ]*
*c)* *[ [1 ,2 ] , [3 ,4, 5] ]*
d) *[ (+) , ( -) , ( *) ]*

**2**. El tipo de la función *uncurry f (x,y) = f x y* es:

*a)* *(a->b->c) -> (a,b) -> c*
*b)* *((a,b) ->c) -> a -> b -> c*
*c)* *a -> b -> c*
*d)* *(a->a) -> (a,a) -> a*

**3**. ¿Cuál de las siguientes expresiones es igual a *[1 ,2 ,3 ,4 ]*?:

*a)* *1:2:3:4*
*b)* *1++2++3++4*
*c)* *map (<=4) [1..]*
d) *take 4 (iterate (+1) 1)*

**4**. ¿Cuál de las siguientes expresiones es igual a *[1, 9, 25]*?

*a)* *[ x^2 | x <- [1,5] ]*
*b)* *[ x^2 | x <- [1..5] ]*
*c)* *map (^2) [1..5]*
*d)* *map (^2) (filter (not.even) [1..6])*

**5.** ¿Cuál de las siguientes definiciones es válida?

*a)* ***data*** *SnocList a = a | SnocList a* ***deriving*** *Show*
*b)* ***data*** *SnocList a = Vacía | Snoc (SnocList a) a* ***deriving*** *Show*
*c)* ***data*** *SnocList a = Vacía | SnocList (Snoc a) a* ***deriving*** *Show*
*d)* ***data*** *SnocList a = Vacía | SnocList (Snoc b) b* ***deriving*** *Show*

**6.** Considérese las siguientes definiciones:

```
data Nat = Z | S Nat   deriving Show
toma 0       x     = x
toma (n+1) (S x) = S (toma n x)
```

¿Cuál es el tipo de *toma*?

a) *toma* no es correcta.
b) *toma :: Nat -> Nat -> Nat*
c) *toma :: Nat -> Integer -> Nat*
d) *toma :: Integer -> Nat -> Nat*

**7.** ¿Cuál es el tipo de la siguiente función?

*agrupa x = ( x, show x, x<=x)*

a) *agrupa :: Eq a => a -> (a,a,a)*
b) *agrupa :: a -> (a,b,c)*
c) *agrupa :: (Show a, Ord a) => a -> (a, String, Bool)*
d) *agrupa :: a -> (a, String, Bool)*

**8.** ¿Cuál de los siguientes valores es un dato de tipo *Árbol Int*?

***data** Árbol a = V | N (Árbol a) a (Árbol a) **deriving** Show*

a) *N (N V) 5 (N V)*
b) *N V 2 ( N V 3 V)*
c) *N (5 V 5)*
d) *V (N V 3 V) 2 V*

**Respuestas:**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

# **INFORMATICA**.

FORTRAN Extraordinario Febrero 2003

**Apellidos, Nombre** | **Grupo**

1. *(4 puntos)* Se dice que un vector de *n* componentes es **mayoritario** si al menos uno de los elementos almacenados en el vector se repite más de *n*/2 veces. Escribe una función que tome como parámetro un vector de elementos enteros y compruebe si es o no es mayoritario.
   **NOTA**: Se valorará la eficiencia de la solución.

2. (6 *puntos*) Se dispone de una serie de números enteros cuyos valores están entre 1 y 100 almacenados en un fichero. En cada línea del fichero hay **dos** números y cada número ocupa un ancho de **tres** caracteres.

   Escribe un programa que lea del teclado el nombre de un fichero de entrada como el anterior y calcule

   a) Cuántos números hay almacenados en el fichero
   b) Cuántos de éstos números son mayores que 50
   c) La suma de todos los números almacenados en el fichero
   d) La media aritmética de todos los números almacenados en el fichero

Los resultados deben mostrarse por pantalla.