UNIVERSIDAD DE MALAGA
DEPARTAMENTO DE LENGUAJES Y
CIENCIAS DE LA COMPUTACION

Examen de P. DECLARATIVA
Septiembre de 2004
Ejercicio de Programación Funcional

1.a (*0.75 puntos*) Define, usando exclusivamente las funciones *map* y *sum*, la función *length* que calcule la longitud de una lista:

```
length = sum . map (\_ -> 1)
```

1.b (*0.75 puntos*) Escribe, usando una lista por comprensión, una función `posiciones` que devuelva una lista con las posiciones en las que aparece un elemento en una lista:

```
posiciones False [True,False,True,False] => [1,3]
```

Da también el tipo polimórfico de ésta función.

```
posiciones :: Eq a => a -> [a] -> [Int]
posiciones x xs = [ i | (x',i) <- zip xs [0..], x == x']
```

1.c (*0.75 puntos*) ¿Cuál es el tipo de la siguiente función?

```
second  = head . tail
```

```
second :: [a] -> a
```

1.d (*0.75 puntos*) Da un ejemplo de uso de la función *second*, que aunque sea correcta con respecto a los tipos produzca un error en tiempo de ejecución.

```
second [1]
```

2 Considera el siguiente tipo para representar árboles generales con información en las hojas:

```
data ArbolGH a = HojaGH a  | NodoGH  [ArbolGH a]
```

2.a *(1 punto)* Define una función veces que cuente las `veces` que aparece un elemento en un árbol.

```
veces :: (Eq a) => a -> ArbolGH a -> Int
veces x (HojaGH y)
  | x == y    = 1
  | otherwise = 0
veces x (NodoGH xs) = sum (map (veces x) xs)
```

2.b *(1 punto)* Define una función foldGH que realice un plegado sobre este tipo de árboles.

```
foldGH :: ([b] -> b) -> (a -> b) -> ArbolGH a -> b
foldGH f g (HojaGH x) = g x
foldGH f g (NodoGH xs) = f (map (foldGH f g) xs)
```

2.c *(1 punto)* Define la función veces utilizando la función foldG anterior .

```
veces' :: (Eq a) => a -> ArbolGH a -> Int
veces' x = foldGH sum (\y -> if x==y then 1 else 0)
```

3.

3.a (*0.75 puntos*) Define de modo recursivo una función `cuentaMax` que dada una lista de números naturales devuelva un par con el máximo elemento y el número de veces que aparece.

```
cuentaMax [1,3,2,3,2,1,1] => (2,3)
```

3.b (*1.25 puntos*) Define la función `cuentaMax` usando `foldr`.

```
cuentaMax = foldr f (0,0)
 where
  f x (n,c)
   | x == n     = (n,c+1)
   | x < n      = (n,c)
   | otherwise  = (x,1)
```

3.c (*0.75 puntos*) Define de modo recursivo una función `reempSuma` que dada una lista de números naturales devuelva un lista donde cada elemento de la original es sustituida por la suma de dicho elementos y los que le suceden en la lista.

```
reempSuma [10,20,30]  => [10+20+30, 20+30, 30] => [60,50,30]
```

3.d (*1.25 puntos*) Define la función `reempSuma` usando `foldr`.

```
reempSuma = fst . foldr (\x (ys,st) -> let y = x+st in (y:ys,y)) ([],0)
```