INFORMATICA.
EXAMEN FINAL Junio 2007. Haskell

Apellidos, Nombre	
DNI	

La puntuación total del examen es 10 puntos. Para aprobar será necesario un mínimo de 5 puntos.

1. Con objeto de escribir un programa para una máquina expendedora, consideremos las siguientes definiciones:

```
type Valores = [Float]
type Monedas = [Integer]
```

Consideremos además que la variable `val` se inicializa del siguiente modo al principio del programa:

```
val :: Valores
val = [ 0.01, 0.05, 0.10, 0.20, 0.50, 1.0, 2.0 ]

tiposDeMonedas = length val
```

La idea es que la máquina expendedora puede manejar siete tipos de monedas, cada uno con los valores indicados (1 céntimo, 5 céntimos, 10 céntimos, 20 céntimos, 50 céntimos, 1 euro y 2 euros), y `val !! i` (con $0 \leq$ `i` ≤ 6) indica el valor en euros del tipo de moneda `i`-ésimo.

El tipo `Monedas` se utiliza para representar una cantidad de dinero indicando el número de monedas de cada clase. Por ejemplo, con:

```
c :: Monedas
c = [1, 0, 0, 0, 1, 2, 0]
```

la cantidad representada es 2.51 euros (1 moneda de 1 céntimo, 1 moneda de 50 céntimos y 2 monedas de 1 euro).

a) (1 punto) Recuerda que la función `zipWith` se encuentra predefinida del siguiente modo:

```
zipWith f (x:xs) (y:ys) = f x y : zipWith f xs ys
zipWith _  _      _     = []
```

Escribe, usando `zipWith`, una función `valor` que tome un parámetro de tipo `Valores` y otro de tipo `Monedas` y devuelva un **`Float`** que se corresponda con el valor en euros.

b) (2 *puntos*) Con objeto de resolver el problema de devolver el cambio de una compra, escribe una función `cambio` que tome un parámetro de tipo `Valores`, otro de tipo **`Float`** que indique la cantidad total en euros introducida por el usuario de la máquina, otro de tipo **`Float`** que indique el precio del producto comprado y devuelva un resultado de tipo `Monedas` que indique el cambio a devolver, de modo que el número total de monedas devuelto sea mínimo. Asume que la máquina dispone de tantas monedas de cada tipo como sea necesario para el cambio.

Por ejemplo:

```
cambio val 10 2.5 => [0, 0, 0, 0, 1, 1, 3]
```

ya que el cambio (7.5 euros) se corresponde a 1 moneda de 50 céntimos, una moneda de 1 euro y 3 monedas de 2 euros).

c) (1 *punto*) Resuelve el problema del apartado **b)**, pero suponiendo que el número de monedas disponibles de cada clase para el cambio está limitado. Para ello, define una función `cambioLimitado`, que además de los parámetros de la función

`cambio`, tome otro de tipo `Monedas` que indique la cantidad disponible de cada tipo de moneda.

d) (1 *punto*) Resuelve el problema del apartado **b)** usando `foldl`.

2. El método de ordenación *Bucket Sort* permite ordenar una lista de valores que están dentro de cierto rango. A modo de ejemplo, supongamos que los valores que queremos ordenar son:

$$[7,\quad 6,\quad 1,\quad 3,\quad 7,\quad 1,\quad 10,\quad 7]$$

Para ordenar estos valores, se crea un *cubo* por cada valor del rango. La idea es que el cubo c_i ha de contar el número de veces que aparece el valor i en la lista que queremos ordenar. Para el ejemplo dado, los cubos calculados serían:

2	0	1	0	0	1	3	0	0	1
C_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}

Una vez calculados los cubos, es fácil obtener la lista ordenada correspondiente. Basta con recorrer los cubos de izquierda a derecha y añadir a la lista de datos ordenados tantos elementos con valor i como valor tenga cada uno de los cubos c_i :

$$[\underbrace{1,\ 1,}_{\text{por } c_1}\ \underbrace{3,}_{\text{por } c_3}\ \underbrace{6,}_{\text{por } c_6}\ \underbrace{7,\ 7,\ 7,}_{\text{por } c_7}\ \underbrace{10}_{\text{por } c_{10}}\ ]$$

a) (3 ptos.) Escribe una función **sobrecargada** en Haskell `bucketSort` que tome como parámetro una lista y la ordene usando el método descrito. Para representar los cubos, usa una lista de pares, donde la primera componente de un par sea un valor y la segunda el nº de veces que aparece. Por ejemplo, para los cubos anteriores, la lista sería:

[(1,2), (2,0), (3,1), (4,0), (5,0), (6,1), (7,3), (8,0), (9,0), (10,1)]

b) (2 ptos.) Sea el siguiente tipo para representar un árbol de búsqueda que contiene un par en cada tupla:

```
data ÁrbolB a = VacíoB
              | NodoB (ÁrbolB a) (a,Int) (ÁrbolB a)
                deriving Show
```

de modo que cada par contiene un valor y el número de veces que aparece en el árbol. Escribe de nuevo la función `bucketSort`, pero representado los cubos como un árbol en vez de cómo una lista de pares.

INFORMATICA.
EXAMEN FINAL Junio 2007. Pascal

Apellidos, Nombre	
DNI	

La puntuación total del examen es 10 puntos. Para aprobar será necesario un mínimo de 5 puntos.

1) Consideremos los siguientes tipos para representar los datos relativos a personas:

```
CONST
     MAX = 100;
TYPE
     PERSONA = RECORD
                 Nombre, Apellido : String;
                 Edad : Cardinal
               END;
     VECTOR = ARRAY [1..MAX] OF PERSONA;
```

a) (2.5 *pts*) Escribe un subprograma que tome como parámetro un `VECTOR` y lo ordene usando el método de **ordenación por inserción** de modo **DESCENDENTE,** es decir de mayor a menor, tomando como criterio de ordenación los apellidos de las personas. Para comparar alfabéticamente dos apellidos, puedes usar los operadores relacionales de Pascal (<, <=, >, etc.).

b) (1.5 *pts*) Escribe una función que tome como parámetro un `VECTOR` de personas ordenado **DESCENDENTE** y un apellido, y realice una búsqueda binaria dentro del vector. Si se encuentra el apellido, la función debe devolver la posición de éste dentro del vector. Si no se encuentra el apellido, la función debe devolver cero.

2) Supongamos que disponemos de una serie de cadenas de caracteres tales que, cada cadena contiene una o más palabras seguidas cada una de un punto. Por ejemplo:

```
'la.casa.roja.'
'el.coche.azul.'
'la.moto.baja.la.cuesta.'
'el.lazo.verde.'
```

a) (2 pts.) Define una función `primeraPalabra` que tome como parámetro una cadena de las anteriores y devuelva una cadena con su primera palabra. Por ejemplo, `primeraPalabra('la.casa.roja.')` debe devolver `'la'`. Observa que la primera palabra son las letras antes del primer punto. Recuerda que la concatenación de cadenas en Pascal se realiza con el operador +, y que la cadena vacía se escribe `''`. Recuerda también que que para accedera al carácter i-ésimo de una cadena se puede usar la notación de arrays.

b) (2 pts.) Supongamos que ya está definida una función

```
Function quitaPrimeraPalabra(l : String) : String;
```

que dada una cadena, devuelve la cadena que se obtiene al eliminar su primera palabra. Por ejemplo, `quitaPrimeraPalabra('la.casa.roja.')` devolverá `'casa.roja.'`.

Usando esta función y la del apartado anterior, define una función `cuenta` que tome como parámetros una palabra y una cadena de palabras y devuelva cuántas veces aparece la palabra en la cadena. Por ejemplo, `cuenta('la','la.moto.baja.la.cuesta')` debe devolver 2.

c) (1 pt.) Da la definición de la función `quitaPrimeraPalabra`.

d) (1 pt.) Define una función `esSubcadena` que devuelva `TRUE` si una cadena es subcadena de otra, es decir, si la primera aparece dentro de la segunda. Por ejemplo, `esSubcadena('la','malaga')` debe devolver `TRUE`, mientras que `esSubcadena('la','maleta')` debe devolver `FALSE`.