Práctica 7
Programación Funcional - Listas

Algoritmos y Estructura de Datos I

Segundo Cuatrimestre 2010

Ejercicio 1. Definir las siguientes funciones sobre listas:

1. *ultimo* :: [*Integer*] → *Integer*, que devuelve el último elemento de una lista no vacía.
2. *principio* :: [*Integer*] → [*Integer*], que recibe una lista no vacía y la devuelve sin su último elemento.
3. *todosIguales* :: [*Integer*] → B*ool*, que dada una lista devuelve verdadero si y solamente si todos sus elementos son iguales.
4. *todosDistintos* :: [*Integer*] → B*ool*, idem inciso anterior, devolviendo verdadero cuando todos los elementos son diferentes.
5. *mismosElementos* :: [*Integer*] → [*Integer*] → *Bool*, que dadas dos listas de enteros devuelve verdadero si y solamente si ambas listas contienen los mismos elementos, sin tener en cuenta repeticiones.

Ejercicio 2. Definir las siguientes funciones sobre listas de caracteres:

1. *sacarTodos* :: [*Char*] → [*Char*] → [*Char*], que elimina todas las apariciones de la primer lista en la segunda. Por ejemplo *sacarTodos* $[c, a]$ $[a, c, a, d, c, a]$ es $[a, d]$.
2. *sacarBlancosRepetidos* :: [*Char*] → [*Char*], que reemplaza cada subsecuencia de blancos contiguos del primer parámetro por un solo blanco en el segundo parámetro.
3. *contarPalabras* :: [*Char*] → *Integer*, que devuelve la cantidad de palabras del parámetro.
4. *palabraMasLarga* :: [*Char*] → [*Char*], que devuelve la palabra más larga del parámetro.
5. *palabras* :: [*Char*] → [[*Char*]], que arma una nueva lista con las palabras del parámetro.
6. *aplanar* :: [[*Char*]] → [*Char*], que a partir de una lista de palabras arma una lista de caracteres concatenándolas.
7. *aplanarConBlancos* :: [[*Char*]] → [*Char*], que a partir de una lista de palabras, arma una lista de caracteres concatenándolas e insertando un blanco entre cada par.
8. *aplanarConNBlancos* :: [[*Char*]] → *Integer* → [*Char*], que a partir de una lista de palabras, arma una lista de caracteres concatenándolas e insertando n blancos entre cada par (n debe ser no negativo).
9. *nat2bin* :: *Integer* → [*Integer*], que recibe un número no negativo y lo transforma en una lista de bits correspondiente a su representación binaria. Por ejemplo *nat2bin* 8 devuelve $[1, 0, 0, 0]$.
10. *bin2nat* :: [*Integer*] → *Integer*, que recibe una lista de bits y la transforma en el número entero no negativo representado por dicha lista. Por ejemplo *bin2nat* $[1, 0, 0, 0, 1]$ devuelve 17.
11. *nat2hex* :: *Integer* → [*Char*], que recibe un número no negativo y lo transforma en una lista de caracteres correspondiente a su representación hexadecimal. Por ejemplo *nat2hex* 45 devuelve $['2', 'D']$.

Ejercicio 3. Diremos que un entero no negativo *a reduce* en *b* (notamos $a \rightsquigarrow b$) sii *b* es el doble de la multiplicación de todos los dígitos de *a*. Por ejemplo $99 \rightsquigarrow 162$, $71 \rightsquigarrow 14$, $237 \rightsquigarrow 84$. Supongamos que reducimos sucesivamente un número n

$$n \rightsquigarrow n_1 \rightsquigarrow n_2 \rightsquigarrow n_3 \rightsquigarrow \dots$$

Sea $n_i = n_j$, con $i < j$, siendo i la primera repetición de esta cadena de reducciones. Definir una función que dado n devuelva i, el número de reducción correspondiente a la primera repetición (en caso de que exista). Por ejemplo, si la función es *nreduc* :: *Integer* → *Integer* , el resultado de *nreduc* 99 es 3 dado que

$$99 \rightsquigarrow 162 \rightsquigarrow 24 \rightsquigarrow \mathbf{16} \rightsquigarrow 12 \rightsquigarrow 4 \rightsquigarrow 8 \rightsquigarrow \mathbf{16} \rightsquigarrow 12 \rightsquigarrow \dots$$

Para *nreduc* 59 el resultado es 2, ya que si reducimos sucesivamente el 59, obtenemos

$$59 \rightsquigarrow 90 \rightsquigarrow \mathbf{0} \rightsquigarrow \mathbf{0} \rightsquigarrow \dots$$

Análogamente para el *nreduc* 237 el resultado es 2

$$237 \rightsquigarrow 84 \rightsquigarrow \mathbf{64} \rightsquigarrow 48 \rightsquigarrow \mathbf{64} \rightsquigarrow \dots$$

Ejercicio 4. Dada una lista de enteros L no vacía, especificar y definir una función que devuelva la sublista de L que más sume (en caso de que haya más de una, devolver cualquiera de ellas). Por ejemplo, si $L = [-1, 3, -5, \mathbf{7}, \mathbf{9}, \mathbf{2}, \mathbf{-3}, \mathbf{6}, -10, 2]$, la función debe devolver $[7, 9, 2, -3, 6]$, que suma 21. Si $L = [\mathbf{2}, -3, -1, 0, 1]$, debe devolver [2], que suma 2.

Ejercicio 5. Dada una lista de α L, definir una función que devuelva:

1. la lista L ordenada ascendentemente
2. la lista L ordenada descendentemente
3. una lista de listas de α tal que cada lista tiene todos sus elementos iguales y longitud igual a la cantidad de apariciones de ese elemento en L, además la lista resultante esta "ordenada" ascendentemente. Por ejemplo, $L = [8, 5, 5, 4, 9, 4, 4, 4, 8]$ la funcion deberá devolver [[4,4,4,4],[5,5],[8,8],[9]].

Ejercicio 6. Más ejercicios sobre listas:

1. *reverso* :: [*Char*] → [*Char*], que devuelve la lista original invertida. Por ejemplo *reverso* $[a, c, b]$ es $[b, c, a]$.
2. *capicua* :: [*Char*] → *bool*, que devuelve verdadero si la lista es capicua. Por ejemplo *capicua* $[a, c, b, b, c, a]$ es $true$, *capicua* $[a, c, b, d, a]$ es $false$.
3. *sumaAcumulada* :: [*Int*] → [*Int*], que devuelve una lista, que posee en la posición i-ésima, la suma acumulada de 0 a i en la lista original. Por ejemplo *sumaAcumulada* $[1, 2, 3, 4, 5]$ es $[1, 3, 6, 10, 15]$.
4. *descomponerEnPrimos* :: [*Int*] → [[*Int*]], que devuelve la lista de listas, que resulta de descomponer en números primos cada uno de los números de la lista original. Por ejemplo *descomponerEnPrimos* $[2, 10, 6]$ es $[[2], [2, 5], [2, 3]]$.