



Universidad
de Huelva



Universidad de Huelva

GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICA 1

EJERCICIOS SOBRE LISTAS EN HASKELL

Autor: Alberto Fernández Merchán

Asignatura: Modelos Avanzados de Computación

1. Introducción

Este trabajo consiste en la realización de cinco ejercicios para familiarizarnos con las funciones básicas sobre listas de *Haskell*. Dichos ejercicios serán:

- **cambia_el_primer** **a b**: Cambia el primero valor de la lista **b** por el valor de **a**.
- **cambia_el_n** **a n b**: Cambia el valor que se encuentra en la posición **n** de la lista **b** por el valor de **a**.
- **get_mayor_abs** **a**: Obtiene el mayor valor, en valor absoluto, de la lista **a**.
- **num_veces** **a b**: Obtiene el número de veces que se repite el valor **a** en la lista **b**.
- **palabras_mayores_n** **n a**: Obtiene un subconjunto de la lista **a** con los elementos con una longitud mayor a **n**.

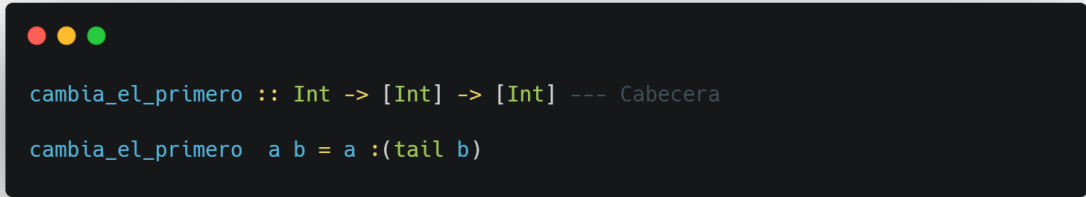
El tipo de datos que se usará para los parámetros de las funciones serán **enteros**, excepto para la última función donde **n** será un entero, pero **a** será una lista de **Strings**.

Finalmente, completaremos la práctica aportando 5 ejemplos que utilicen las funciones vistas. Se incluyen, junto a este documento, un fichero que contiene la cabecera y la declaración de cada una de las funciones propuestas. A continuación se explicará como funciona cada una de ellas.

2. Ejercicios

2.1. Cambia el Primero

En este primer ejercicio debemos cambiar el primer valor de la lista **b** por el valor del parámetro **a**. En la **cabecera** de la función indicamos que la función tiene como entrada un **entero** y una **lista de enteros** y, como salida, devuelve una **lista de enteros**.



```
cambia_el_primer :: Int -> [Int] -> [Int] --- Cabecera
cambia_el_primer a b = a :(tail b)
```

Figura 1: Código de `cambia_el_primer`

Para cambiar el primer valor de la lista aplicamos la función **tail** a una lista para devolver todos los elementos excepto el primero, después le concatenamos el elemento **a** por la izquierda al resultado y obtenemos el elemento **a** concatenado con todos los elementos de la lista **b** menos el primero.

2.2. Cambia el N

Este ejercicio es un poco más complejo que el anterior. Consiste en sustituir el elemento que se encuentre en la posición **N** de la lista **b** por el valor de **a**.

En la cabecera de la función indicamos que los parámetros de entrada serán un entero (**a**), una lista de enteros (**b**) y otro entero (**n**) y, como salida, obtendremos una lista de enteros. Podemos dividir la idea de

```
cambia_el_n :: Int -> [Int] -> Int -> [Int] --- Cabecera
cambia_el_n a b n = take (n) b ++ [a] ++ reverse (take (length b - (n+1)) (reverse b))
```

Figura 2: Código de cambia_el_n

esta función en tres partes:

1. **Obtención de los n primeros elementos:** Para obtener los primeros n elementos tan solo debemos realizar un *take (n) b* que obtiene los n primeros elementos de la lista b.
2. **Adición del valor nuevo:** A continuación debemos concatenar el elemento a, pero como es por la derecha debemos meterlo en una lista. *++[a]*.
3. **Obtención de los últimos elementos:** Los elementos que nos interesarán serán los $(longitud_b - (n + 1))$ de la misma lista dada la vuelta. Una vez los obtenemos la volvemos a invertir y tendríamos los elementos que tenemos que concatenar con la lista anterior. $(reverse(take(length\ b - (n + 1))(reverse\ b)))$

2.3. Obtener Mayor Absoluto

Este ejercicio consiste en obtener el mayor valor de una lista en valor absoluto.

En la cabecera indicaremos que tiene como parámetro de entrada una lista de enteros y como salida un número entero.

```
get_mayor_abs :: [Int] -> Int --- Cabecera
get_mayor_abs a = maximum (map abs a)
```

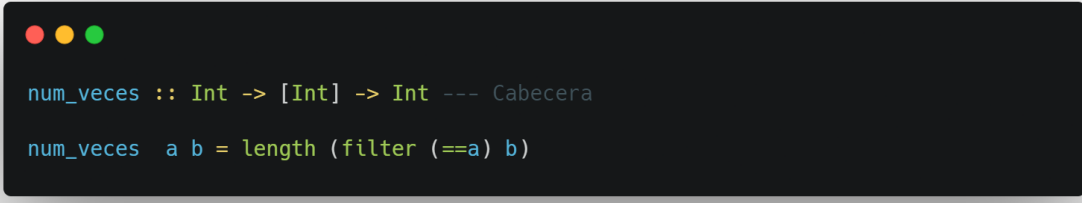
Figura 3: Código de get_mayor_absoluto

La idea de este ejercicio es utilizar la función *map* para aplicarle la operación de valor absoluto a todos los elementos de la lista. Una vez obtenidos todos los valores positivos de la lista, tan solo tenemos que aplicarle la función de obtener el máximo de una lista (*maximum*).

2.4. Número de Veces

Este ejercicio consiste en obtener el número de veces que se repite un valor en una lista. En la cabecera indicaremos un número entero que indica el valor que queremos comprobar, una lista de números enteros y, como salida, un entero que indicará el número de veces que se repite dicho valor.

La idea principal es la de utilizar la función *filter* junto con la función de comparación. De esta forma



```
num_veces :: Int -> [Int] -> Int --- Cabecera  
num_veces a b = length (filter (==a) b)
```

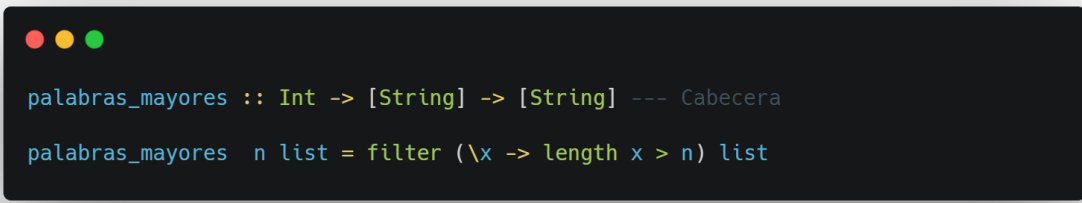
Figura 4: Código de num_veces

podemos obtener un subconjunto de la lista **b** que solo contenga los elementos iguales al valor que queremos comprobar (**a**). Por último le calculamos la longitud a dicha lista y obtendremos el número de veces que se repite el valor en la lista.

2.5. Palabras mayores que N

Por último, este ejercicio consiste en encontrar, en una lista pasada por parámetro, las palabras que contengan un número de caracteres superior a **n**.

En la cabecera indicaremos como parámetros de entrada un entero (**n**) y una **lista de Strings** y, como salida, devolveremos una **lista de Strings**.



```
palabras_mayores :: Int -> [String] -> [String] --- Cabecera  
palabras_mayores n list = filter (\x -> length x > n) list
```

Figura 5: Código de palabras_mayores

Utilizaremos la función *filter* junto a una expresión lambda que permite filtrar los elementos de la lista en función de la longitud de cada uno de sus términos. Dicha función se puede leer de forma que se filtren todos los elementos *x* de la lista, tal que su longitud sea mayor que **n**.

3. Ejemplos


En esta sección indicaremos 5 nuevos ejemplos donde se utilicen las funciones que hemos visto durante esta práctica. Los ejemplos que se plantean son los siguientes:

1. Multiplicar todos los números positivos de una lista.
2. Detectar si una cadena es un palíndromo.
3. Obtener el primero número par de una lista.
4. Obtener la posición de un número en una lista.
5. Sumar una lista de números si se cumple una condición.

3.1. Multiplicación Positiva

Este ejercicio consiste en, dada una lista de números enteros, multiplicar todos los positivos.

La cabecera de esta función indica el parámetro de entrada (lista de enteros) y la salida como un entero.

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in Haskell and defines a function named mult_positiva. The first line is the type signature: mult_positiva :: [Integer] -> Integer. The second line is the function definition: mult_positiva a = foldl (\x y -> x * y) 1 (filter (> 0) a).

```
mult_positiva :: [Integer] -> Integer
mult_positiva a = foldl (\x y -> x * y) 1 (filter (> 0) a)
```

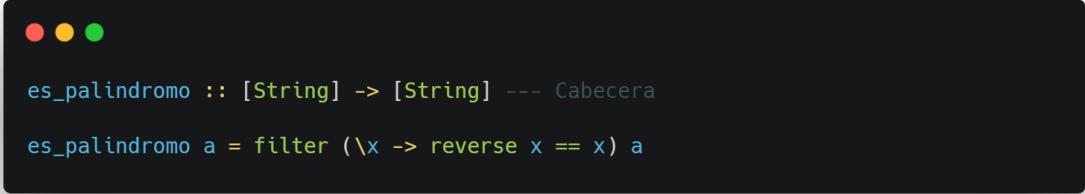
Figura 6: Código de mult_positiva

La idea consiste en utilizar la función *foldl* que aplica una fórmula a un par de valores y los va almacenando en el primero. En este ejemplo multiplicamos el valor acumulado por cada valor de la lista de enteros. Debemos inicializar el valor acumulado a 1 ya que es el elemento neutro de la multiplicación.

3.2. Palíndromo

Este ejercicio consiste en comprobar si una cadena es un palíndromo o no.

La cabecera de esta función indica como parámetro de entrada una lista de String y, como salida, un subconjunto de la anterior lista donde aparecen las palabras que eran palíndromos.



```
es_palindromo :: [String] -> [String] --- Cabecera
es_palindromo a = filter (\x -> reverse x == x) a
```

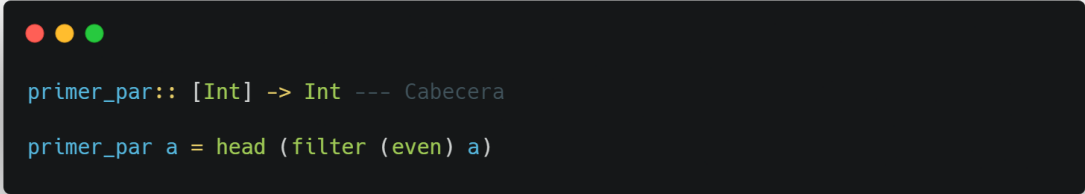
Figura 7: Código de es_palindromo

La idea es comprobar si, cada elemento de la lista de String, es igual a dicho elemento invertido. Si es cierto lo consideramos un palíndromo y supera el filtro de salida.

3.3. Primer número par

Este ejercicio consiste en, dada una lista de enteros, devolver el primer número par que se encuentre.

La cabecera de esta función tiene como parámetro de entrada una lista de enteros y, como salida, un entero.



```
primer_par :: [Int] -> Int --- Cabecera
primer_par a = head (filter (even) a)
```


Figura 8: Código de primer_par

La idea de este ejercicio es sacar el primer número que supere el filtro de los números pares de la lista *a* con la función *head*.

3.4. Obtener posición

Este ejercicio consiste en, dada una lista de números y un número contenido en esa lista, obtener la posición de dicho número.

La cabecera de esta función tiene como parámetros de entrada una lista de enteros y un entero que es el que va a buscar. Como parámetro de salida tiene un entero que indica la posición del número.



```
get_pos::[Int]->Int->Int --Cabecera
get_pos a n = length a - length(dropWhile(\x -> x /= n) a)
```

Figura 9: Código de get_pos

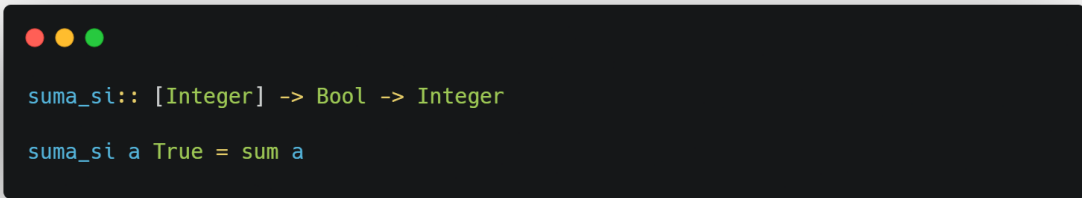
Para conseguir la posición de un elemento debemos restar la longitud de la lista menos la longitud de la sublista que hay a partir de dicho elemento.

Por ejemplo: $a = [1, 2, 3, 4, 5, 6]; n = 4 \rightarrow 6 - \text{length}[4, 5, 6] = 6 - 3 = 3$

3.5. Suma Condicional

Esta función suma todos los números de una lista si se cumple una condición.

La cabecera de esta función indica dos parámetros de entrada (un booleano y una lista de enteros) y uno de salida (un entero).



```
suma_si:: [Integer] -> Bool -> Integer
suma_si a True = sum a
```

Figura 10: Código de suma_si

La suma condicional funciona solamente cuando la condición que se le pasa por parámetro es verdadera. Para ello debemos establecer el parámetro *booleano* a *True* en la declaración de la función.