

LABORATORIO 004

TEMA: Listas, funciones y recursividad en Haskell


I. COMPETENCIAS

Durante el desarrollo de la práctica el estudiante:

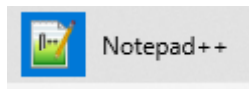
- Define adecuadamente las funciones y uso de listas en Haskell.
- Carga las funciones en el Haskell y las ejecuta.

II. REQUISITOS DE DESARROLLO

Para realizar las prácticas el estudiante debe:

- Conocer y comprender cómo se definen las funciones en el Haskell.
- Conocer el manejo básico del L. P. Haskell en su implementación  WinGHCi y cómo se cargan archivos “.hs” en Haskell.

OBS: Para generar los archivos “*.hs”, utilizar:



III. EJEMPLOS

Digitar en el WinGHCi los ejemplos vistos sobre las funciones en Haskell.

1. Ingresar una lista formando por un conjunto de listas, se debe devolver una lista formado por números enteros de las longitudes correspondiente a cada lista

`[[3],[4,5],[4,5,9]] ---> [1,2,3]`

```
longi :: [[a]] -> [Int]
longi [] = []
longi (x:xs) = length x : longi xs
```

2. Ingresar dos listas de números enteros de la misma cantidad de elementos, mostrar una lista de pares ordenados (max, min), en el cual max. y min representan el mínimo y máximo valor entre las listas ingresar correspondiente a la misma posición

[2,9,4] [6,7,8] ---> [(6,2), (9,7), (8,4)]

```
par :: [Integer] -> [Integer] -> [(Integer,Integer)]
par [] [] = []
par (x:xs) (y:ys) = if (x > y) then (x,y) : par xs ys
else (y,x) : par xs ys
```

3. Ingresar dos listas formados por números enteros, se debe mostrar una sola lista formado por el máximo número entero de la misma posición de las dos listas

[2,9,4] ----> [6,7,8] ----> [6,9,8]

```
zmax :: [Integer] -> [Integer] -> [Integer]
zmax [] [] = []
zmax (x:xs) ([]) = x:xs
zmax ([]) (y:ys) = y:ys
zmax (x:xs) (y:ys) = if x>y then x : zmax xs ys
else y : zmax xs ys
```

4. Se tiene una lista de números enteros, se debe mostrar solamente una lista con números positivos

```
*Main> fpositivos [1,2,-3]
[1,2]
```

```
fpositivos :: [Integer]->[Integer]
fpositivos [] = []
fpositivos (x:xs) = if x>=0 then x : fpositivos xs
else fpositivos xs
```

IV. ACTIVIDAD

PROGRAMACIÓN LÓGICA FUNCIONAL

BLOQUE 01:

- Construye la función $f(x) = 2x$ en un archivo y cárgala en el intérprete. (Comando **:l**).
- Construye la función $g(x, y) = x^2 + y^2$ y recarga el archivo de texto. (Comando **:r**).
- Construye la función $h(x, y) = 6x + x^2 + y^2$ utilizando las funciones anteriores.
- Construye la función:

$$f(x) = \begin{cases} x & \text{si } x > 100 \\ x^2 & \text{si } x \leq 100 \end{cases}$$

utilizando la construcción **if ... then ... else...**

```
--Bloque1
func1 :: Double -> Double
func1 x = 2*x

--
func2 :: Double -> Double -> Double
func2 x y = (x^2) + (y^2)

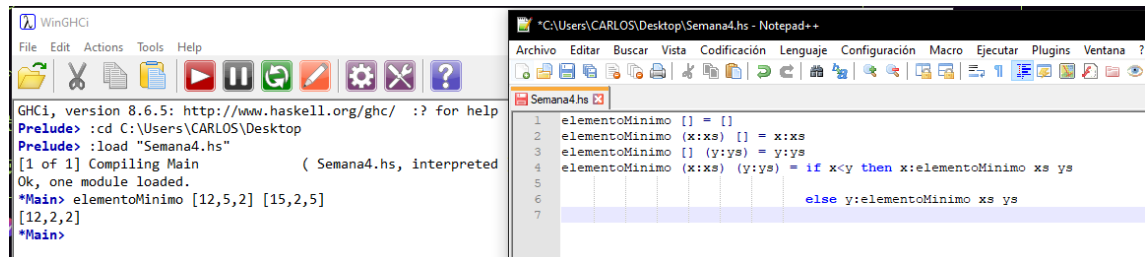
--
func3 :: Double -> Double -> Double
func3 x y = (6*x) + (x^2) + (y^2)

--
func4 :: Double -> Double
func4 x = if x <= 100 then x^2 else x
```

```
*Main> func1 2
4.0
*Main> func2 2 3
13.0
*Main> func3 2 3
25.0
*Main> func4 100
10000.0
*Main> func4 101
101.0
```

BLOQUE 02:

1. Ingresar dos listas formados por números enteros, se debe mostrar una sola lista formado por el mínimo número entero de la misma posición de las dos listas



The screenshot shows two windows. On the left is the WinGHCi window, which is the Haskell interpreter. It shows the command prompt where the user has loaded a file named 'Semana4.hs'. The output shows that the module was loaded successfully. The user then enters the command `elementoMinimo [12,5,2] [15,2,5]`, and the interpreter returns `[12,2,2]`. On the right is a Notepad++ window showing the source code of the `Semana4.hs` file. The code defines the `elementoMinimo` function as follows:

```
1 elementoMinimo [] = []
2 elementoMinimo (x:xs) [] = x:xs
3 elementoMinimo [] (y:ys) = y:ys
4 elementoMinimo (x:xs) (y:ys) = if x < y then x:elementoMinimo xs ys
5                                     else y:elementoMinimo xs ys
```

2. Se tiene una lista de números enteros, se debe mostrar solamente una lista con números positivos y negativos (quitar los ceros)

```
funsc [] = []
funsc (x:xs) = if x /= 0 then x:funsc xs
              else funsc xs
```

```
*Main> funsc [1,0,(-1), 1222,0, 1255]
[1,-1,1222,1255]
```

3. Ingresar una lista de números reales, devolver una lista del doble de cada real ingresado

```
[2.3, (-4.6),4.1] ---> [4.6,-9.2,8.2]
```

```
--Ejercicio03Bloque2
fprod :: [Float] -> [Float]
fprod [] = []
fprod (x:xs) = 2*x : fprod xs
```

```
*Main> fprod [2.3,(-4.6),4.1]
[4.6,-9.2,8.2]
```

4. Se tiene una lista de listas y un número natural n, mostrar solamente las listas que consta de más de n elementos
[[3], [4,5], [4,5,9]] 1---> [[4,5], [4,5,9]]

```
--Ejercicio04Bloque2
listdlis [] = []
listdlis (x:xs) = if length x > 1 then x : listdlis xs
                  else listdlis xs
```

```
*Main> listdlis [[1],[1,2],[1,2,3]]
[[1,2],[1,2,3]]
```

5. Se tiene una lista xs y un número entero n, mostrar una lista con los primeros n elementos de lista ingresada. Pero si lista que se recibe tuviera menos de n elementos, en ese caso se devuelve la lista completa

[5,7,8] 3 ---> [5,7,8]

[5,7,8] 2 ---> [5,7]

```
lista n [] = []
```

```
lista n(x:xs) = take n(x:xs)
```

```
*Main> lista 2[2,3,4]
```

```
[2,3]
```

```
*Main> lista 3[2,3,4]
```

```
[2,3,4]
```

```
...
```

```
*Main> lista 1[]
```

```
[]
```

6. Ingresando una lista y un número natural n, mostrar una lista sin los primeros n elementos que corresponden a la lista recibida. Pero si la lista recibida tiene menos de n elementos devolverá una lista completamente vacía.

[5,7,8] 2 ---> [8]

[5,7,8] 3 ---> []

[5,7,8] 1 ---> [7,8]

```
lista2 n[] = []
```

```
lista2 n(x:xs) = drop n(x:xs)
```

```
*Main> lista2 3[]
```

```
[]
```

```
*Main>
```

```
*Main> lista2 2[5,7,8]
```

```
[8]
```

```
*Main> lista2 3[5,7,8]
```

```
[]
```

```
*Main> lista2 3[5,7]
```

```
[]
```

```
*Main> lista2 1[5,7,8]
```

```
[7,8]
```

ANEXOS:

```
Prelude> lis
[1,2,3,4,5,6]
Prelude> head lis
1
Prelude> lis = [1,2,3,4,5,6]
Prelude> lis
[1,2,3,4,5,6]
Prelude> head lis
1
Prelude> tail lis
[2,3,4,5,6]
```

```
Prelude> lis
[1,2,3,4,5,6]
Prelude> last lis
6
Prelude> init lis
[1,2,3,4,5]
```

```
Prelude> lis
[1,2,3,4,5,6]
Prelude> reverse lis
[6,5,4,3,2,1]
```

```
Prelude> lis
[1,2,3,4,5,6]
Prelude> take 3 lis
[1,2,3]
Prelude> lis
[1,2,3,4,5,6]
Prelude> drop 3 lis
[4,5,6]
Prelude> lis
[1,2,3,4,5,6]
Prelude> maximum lis
6
Prelude> minimum lis
1
```

```
Prelude> lis
[1,2,3,4,5,6]
Prelude> sum lis
21
Prelude> product lis
720
```

```
Prelude> lis
[1,2,3,4,5,6]
Prelude> 8 `elem` lis
False
Prelude> 3 `elem` lis
True
```

```
tamano :: [[c]]->[Int]
tamano []=[]
tamano (x:xs)=length x : tamano xs

*Main> tamano [[2,3],[2,3,4,5],[3]]
[2,4,1]
*Main> tamano [[2,3],[2,3,4,5]]
[2,4]
```
