Práctico Nº 2: Haskell

Ejercicio Nº1: Evalúe las siguientes funciones

1. (min (max 3 4) (max 7 3)) //Aplican solo a dos argumentos
2. (succ 9) +(max 5 4) +1
3. (max (succ (max 6 8)) (succ (min 6 8)))
4. (**div** 15 4)

//Funciones sobre listas. Analizadores de listas: head - tail - last - elem

5. (**head** [3,6,9])
6. `(head ['a','b','c'])`
7. (head “abcd”)
8. (**tail** (head (tail ["ab","cd"])))
9. impares = ["uno", "tres"]
10. (”cinco” : impares) // **:** Constructor de lista
11. ([“siete”] ++ impares) // **++** Constructor de lista. Concatena
12. impares
13. (**last** impares)
14. uno = [3,1,8,5,4,2]
15. dos = [7,9,3,5,1]
16. nueva = [(head uno), (head dos)]
17. (splitAt 3 uno) // Retorna una **tupla**
18. (uno !! 2) // Selecciona el tercer elem de la lista (posición desde cero)
19. (dos !! 0)
20. lista = **[**(uno !! 2), (dos !! 3)**]**
21. (**sum** uno)
22. (product dos)
23. (**null** impares)
24. (reverse impares)
25. (take 2 uno)
26. (drop 2 dos)
27. (**maximum** dos)
28. (minimum impares)
29. (**elem** “cinco” impares)
30. (elem “seis” impares)

// Evaluaciones con operadores lógicos

31. (not (elem “tres” impares))
32. (notElem “tres” impares)
33. ((elem “uno” impares) && (elem “siete” impares))
34. ((elem “uno” impares) || (elem “siete” impares))
35. (length [“Jose”, “Antonio”,”Mario”])
36. (length [[“Jose”, “Antonio”,” Mario”]])

Práctico Nº 2: Haskell

Ejercicio Nº2: Tuplas

1. (fst (“ana”,”carlos”)) --Aplica solo duplas, no sobre triplas, cuadruplas, etc.
2. (snd (“ana”,”carlos”)) --Aplica solo duplas
3. (zip [1..] ["uno”, ['d','o','s'],"tres","cuatro"]) – Aplica solo a dos argumentos

Ejercicio N°3: Rangos y listas infinitas. Evalúe las siguientes funciones

1. abecedario = ['a'..'z'])
2. pares = [2, 4..20]
3. impares = [21, 19..1]
4. (take 10 [11, 22..])
5. (take 10 (cycle [‘a’,’b’,’c’]))
6. (take 10 (repeat 10))
7. (replicate 10 10)

Ejercicio N° 4: Listas intencionales. Evaluar las siguientes expresiones:

1. **[**x*2 **|** x **<-** [1..10]**]**
2. **[**x*2 **|** x **<-** [1..10], x*2 >= 12**]**
3. frutas = [“naranjas”, “peras”, “uvas”, “mandarinas”, “peras”]
4. [x | x <- frutas, x=="peras"]
5. lista= [3, 5, 2, 0, 4, 0, 1, 5, 0]
6. sum [1 | x <- lista, x==0]
7. length [x | x <- lista, x/=0]

Ejercicio N°5: Definir una función que cuente los elementos pares de una lista de números.

Ejercicio Nº 6: Definir una función que reciba una lista de listas y entregue la cantidad de elementos de la lista de mayor longitud.

Ejercicio Nº 7: Definir una función que transforme una lista de números en otra lista que contenga el cubo de cada elemento.

Ejercicio Nº 8: Definir una función recursiva que permita eliminar los elementos repetidos de una lista de átomos.

Ejercicio Nº 9: Implementar una función recursiva que pase un número decimal a binario

Ejercicio Nº 10: Implementar una función recursiva que permita obtener la unión de dos listas dadas; los elementos repetidos solo deben aparecer una vez.

Ejercicio Nº 11: Construir un programa no recursivo que realice la suma de números complejos, los cuales se ingresan en sublistas con pares de números donde el primer elemento es la componente real y el segundo la componente imaginaria.

Ejercicio Nº 12: Dada una lista ordenada y un átomo escribir una función que inserte el átomo en el lugar correspondiente

Ejercicio Nº 13: Calcular la suma de dos matrices.

Práctico Nº 2: Haskell

Ejercicio Nº 14: Analizar las siguientes construcciones, hacer el seguimiento y escribe resultado:

1. zipWith (++) [[1, 2], [3, 4]] [[5, 6], [7, 8]]
2. [**[**x, y**]** | x <- [1, 2], y <- [10, 20]]
3. Sea lista = [[1, 3], [2, 5, 6], [4,7]]

   [x **|** xs **<-** lista, x <- xs**]**

4. **[(**x, y**) |** x **<-** ["Ana", "Juan", "Carlos"], y **<-** [4,3]**]**

Ejercicio Nº 15: Dada la siguiente función:

```
buscar::[[Char]] -> [[Char]]
buscar [ ] = [ ]
buscar (x:xs) = [p | p <- x, elem p ['a'..'z']] : buscar xs
```

Realizar el seguimiento para: **buscar ["Estamos", "aprendiendo", "HasKell."]**

Ejercicio Nº 16: Hacer el seguimiento de la siguiente función y decir que entrega para el ejemplo propuesto: Ej: [[1, 2, 3], [2, 3, 4, 5], [6, 7, 8]]

```
pp::(Integral a) => [[a]] -> [a]
pp [] = []
pp (x:xs) = [head x] ++ pp  xs
```

PROPUESTOS

Propuesto Nº 1: Defina una función que, aplicada a una lista de listas, permita obtener una lista de un solo nivel.

Propuesto Nº 2: Sea lista1 = [[100, 1], [200, 2]] y lista2 = [[2, 20], [1, 10]]

Evaluar la siguiente construcción:

[(head x) **:** [last y | y <- lista2, last x == head y] **|** x **<-** lista1**]**

Propuesto Nº 3: Evaluar la siguiente construcción:

map **(**x **->** map (**\y ->** (x , y)) ['a', 'b']**)** [1..3]

Propuesto Nº 4: Realizar el seguimiento para **scanner 3 [2,5,1]**

```
scanner :: (Integral a)-> a -> [a] -> [a]
scanner n [] = [n]
scanner n (x:xs) = x + head (scanner n xs) : scanner n xs
```

Propuesto Nº 5: Escriba un programa que recibiendo como argumento una lista de listas donde cada sublista contiene nombre del docente, dedicación y carrera donde trabaja; entregue como resultado una lista con los nombres de los docentes que cobrarán un plus considerando que los cobrarán aquellos docentes que tenga solamente un cargo con dedicación simple.

Ejemplo:

plus [["Ana","Exclusivo","LSI"],["Mary","Semi","LCC"],["Jose","Simple","LSI"], ["Mary","Simple","LSI"], ["Pepe","Simple","LSI"],…..]

[“Jose”, “Pepe”, ….]