Instituto Tecnológico Superior del Sur de Guanajuato

Programación Lógica Funcional

Ejercicios en Haskell 2

Profesor: Gustavo Iván Vega

Alumno: Cristobal Sánchez Orduña

Informe

1) Desarrollar una función que simule una calculadora científica que permita calcular el seno, coseno, tangente, exponencial y logaritmo neperiano. La función preguntará al usuario el valor y la función a aplicar, y mostrará por pantalla una tabla con los enteros de 1 al valor introducido y el resultado de aplicar la función a esos enteros.

Código:

```haskell
import Text.Printf (printf)


seno :: Double -> Double
seno x = sin x

coseno :: Double -> Double
coseno x = cos x

tangente :: Double -> Double
tangente x = tan x

exponencial :: Double -> Double
exponencial x = exp x

logNeperiano :: Double -> Double
logNeperiano x = log x

mostrarTabla :: (Double -> Double) -> Double -> IO ()
mostrarTabla funcion valor = do
    putStrLn "Entero | Resultado"
    putStrLn "-------------------"
    mapM_ (\x -> printf "%6.0f | %10.6f\n" x (funcion x)) [1.0,2.0..valor]

main :: IO ()
main = do
    putStrLn "Calculadora Científica"
    putStrLn "1. Seno"
    putStrLn "2. Coseno"
    putStrLn "3. Tangente"
    putStrLn "4. Exponencial"
    putStrLn "5. Logaritmo Neperiano"   _ :: String
    putStrLn "Seleccione la función a aplicar (1-5):"
    opcion <- getLine
    putStrLn "Ingrese el valor:"
    valor <- getLine
    let funcion = case opcion of
                    "1" -> seno
                    "2" -> coseno
                    "3" -> tangente
                    "4" -> exponencial
                    "5" -> logNeperiano
                    _   -> error "Opción no válida"
    putStrLn "Tabla de resultados:"
    mostrarTabla funcion (read valor)
```

```
PS C:\Users\crili\OneDrive\Escritorio\Crist
Calculadora Científica
1. Seno
2. Coseno
3. Tangente
4. Exponencial
5. Logaritmo Neperiano
Seleccione la función a aplicar (1-5):
3
Ingrese el valor:
5
Tabla de resultados:
Entero | Resultado
------------------
     1 |   1.557408
     2 |  -2.185040
     3 |  -0.142547
     4 |   1.157821
     5 |  -3.380515
PS C:\Users\crili\OneDrive\Escritorio\Crist
```

2) Desarrollar una función que reciba otra función booleana y una lista, y devuelva otra lista con los elementos de la lista que devuelvan True al aplicarles la función booleana.

Código:

Ejercicio2.hs > main

```haskell
filtrarPorFuncion :: (a -> Bool) -> [a] -> [a]
filtrarPorFuncion _ [] = []
filtrarPorFuncion funcion (x:xs)
    | funcion x = x : filtrarPorFuncion funcion xs
    | otherwise = filtrarPorFuncion funcion xs

main :: IO ()
main = do
    let listaOriginal = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    putStrLn "Lista original:"
    print listaOriginal
    putStrLn "Lista filtrada por ser mayor que 5:"
    print (filtrarPorFuncion (> 5) listaOriginal)
```

Salida:

```
PS C:\Users\crili\OneDrive\Escritorio\Cris
Lista original:
[1,2,3,4,5,6,7,8,9,10]
Lista filtrada por ser mayor que 5:
[6,7,8,9,10]
```

3) Escribir una función reciba una lista de calificaciones y devuelva la lista de calificaciones correspondientes a esas notas. 95-100(excelente), 85-94(Notable), 75-84(Bueno), 70-74(Suficiente) <70(Desempeño insuficiente).

Código:

Ejercicio3.hs > asignarCalificacion

```haskell
data Calificacion = A | B | C | D | F deriving (Show)

asignarCalificacion :: Double -> Calificacion
asignarCalificacion nota
    | nota >= 90 = A
    | nota >= 80 = B
    | nota >= 70 = C
    | nota >= 60 = D
    | otherwise  = F

asignarCalificaciones :: [Double] -> [Calificacion]
asignarCalificaciones = map asignarCalificacion

main :: IO ()
main = do
    let notas = [85, 95, 60, 75, 45]
    putStrLn "Lista de notas:"
    print notas
    putStrLn "Lista de calificaciones:"
    print (asignarCalificaciones notas)
```

Salida:

```
PS C:\Users\crili\OneDrive\Escritorio
Lista de notas:
[85.0,95.0,60.0,75.0,45.0]
Lista de calificaciones:
[B,A,D,C,F]
```

4) Escribir una función reciba un diccionario con las asignaturas y las notas de un alumno y devuelva otro diccionario con las asignaturas en mayúsculas y las calificaciones correspondientes a las notas aprobadas. 95-100(excelente), 85-94(Notable), 75-84(Bueno), 70-74(Suficiente) <70(Desempeño insuficiente).

Código:

Ejercicio4.hs > main

```haskell
import qualified Data.Map as Map

data Calificacion = Excelente | Notable | Bueno | Suficiente | Insuficiente deriving (Show)

asignarCalificacion :: Double -> Calificacion
asignarCalificacion nota
    | nota >= 95 = Excelente
    | nota >= 85 = Notable
    | nota >= 75 = Bueno
    | nota >= 70 = Suficiente
    | otherwise  = Insuficiente

procesarNotas :: Map.Map String Double -> Map.Map String Calificacion
procesarNotas = Map.map (asignarCalificacion)

main :: IO ()
main = do
    let notas = Map.fromList [("Matematicas", 92), ("Historia", 78), ("Fisica", 64), ("Arte", 88)]
        notasProcesadas = procesarNotas notas
    putStrLn "Notas originales:"
    print notas
    putStrLn "Notas procesadas:"
    print notasProcesadas
```

Salida:

```
PS C:\Users\crili\OneDrive\Escritorio\Cristobal Sanchez Orduña - Portafolio\Unidad 2\Ejercicios e
Notas originales:
fromList [("Arte",88.0),("Fisica",64.0),("Historia",78.0),("Matematicas",92.0)]
Notas procesadas:
fromList [("Arte",Notable),("Fisica",Insuficiente),("Historia",Bueno),("Matematicas",Notable)]
```

5) Una inmobiliaria de una ciudad maneja una lista de inmuebles como la siguiente:

[{'año': 2000, 'metros': 100, 'habitaciones': 3, 'garaje': True, 'zona': 'A'},
{'año': 2012, 'metros': 60, 'habitaciones': 2, 'garaje': True, 'zona': 'B'},
{'año': 1980, 'metros': 120, 'habitaciones': 4, 'garaje': False, 'zona': 'A'},
{'año': 2005, 'metros': 75, 'habitaciones': 3, 'garaje': True, 'zona': 'B'},
{'año': 2015, 'metros': 90, 'habitaciones': 2, 'garaje': False, 'zona': 'A'}]

Construir una función que permita hacer búsqueda de inmuebles en función de un presupuesto dado. La función recibirá como entrada la lista de inmuebles y un precio, y devolverá otra lista con los inmuebles cuyo precio sea menor o igual que el dado. Los inmuebles de la lista que se devuelva deben incorporar un nuevo par a cada diccionario con el precio del inmueble, donde el precio de un inmueble se calcula con la siguiente fórmula en función de la zona:

Zona A: precio = (metros * 1000 + habitaciones * 5000 + garaje * 15000) * (1-antiguedad/100)

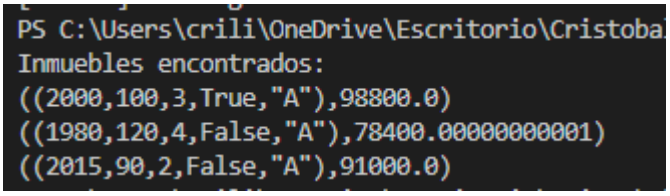
Zona B: precio = (metros * 1000 + habitaciones * 5000 + garaje * 15000) * (1-antiguedad/100) * 1.5

Código:

Ejercicio5.hs > main

```haskell
type Inmueble = (Int, Int, Int, Bool, String)
type Precio = Double

precioInmueble :: Inmueble -> Precio
precioInmueble (año, metros, habitaciones, garaje, zona)
    | zona == "A" = precio * (1 - fromIntegral (2024 - año) / 100)
    | zona == "B" = precio * (1 - fromIntegral (2024 - año) / 100) * 1.5
    where
        precio = fromIntegral (metros * 1000 + habitaciones * 5000 + if garaje then 15000 else 0)

buscarInmuebles :: [Inmueble] -> Precio -> [(Inmueble, Precio)]
buscarInmuebles inmuebles presupuesto = map agregarPrecio $ filter (\inmueble -> precioInmueble inmueble <= presupuesto) inmuebles
    where
        agregarPrecio inmueble = (inmueble, precioInmueble inmueble)

-- Ejemplo de uso
inmuebles :: [Inmueble]
inmuebles = [
    (2000, 100, 3, True, "A"),
    (2012, 60, 2, True, "B"),
    (1980, 120, 4, False, "A"),
    (2005, 75, 3, True, "B"),
    (2015, 90, 2, False, "A")
    ]

main :: IO ()
main = do
    let presupuesto = 100000
        inmueblesFiltrados = buscarInmuebles inmuebles presupuesto
    putStrLn "Inmuebles encontrados:"
    mapM_ print inmueblesFiltrados
```

Salida:


PS C:\Users\crili\OneDrive\Escritorio\Cristoba
Inmuebles encontrados:
((2000,100,3,True,"A"),98800.0)
((1980,120,4,False,"A"),78400.00000000001)
((2015,90,2,False,"A"),91000.0)