Guía de Laboratorio: Functores y Aplicaciones de Listas

Programación Lógica y Funcional
Universidad Tecnológica del Perú

Docente: Carlos R. P. Tovar

Objetivos de la Sesión

- Comprender el concepto de functor en Haskell
- Aplicar functores para transformar datos en contextos
- Utilizar funciones de orden superior con listas
- Implementar ejemplos prácticos de functores con listas
- Desarrollar habilidades para procesamiento de datos con functores

Material Requerido

- GHC (Glasgow Haskell Compiler) instalado
- Editor de texto con soporte para Haskell
- Esta guía de laboratorio

Ejercicio 1: Introducción a Functores (20 min)

Implementa las instancias de Functor para tipos personalizados y utiliza fmap.

```haskell
-- Tipo Maybe ya tiene instancia de Functor
ejemploMaybe :: Maybe Int -> Maybe Int
ejemploMaybe = fmap (+1)

-- Tipo Either ya tiene instancia de Functor
ejemploEither :: Either String Int -> Either String Int
ejemploEither = fmap (*2)

-- Crea tu propio tipo y su instancia de Functor
data Resultado a = Exito a | Error String deriving Show

instance Functor Resultado where
    fmap f (Exito x) = Exito (f x)
    fmap _ (Error msg) = Error msg

-- Ejemplos de uso:
-- fmap (+1) (Exito 5) -> Exito 6
-- fmap (+1) (Error "problema") -> Error "problema"
```

Tareas:

1. Completa la instancia de Functor para Resultado
2. Crea una función que transforme una lista de Resultados
3. Implementa una función que aplique una transformación a un Resultado sólo si es Exito

Ejercicio 2: Aplicaciones de Listas con Map y Filter (25 min)

Trabaja con funciones de orden superior para procesar listas.

```
-- Datos de ejemplo
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

noms = ["Ana", "Carlos", "Elena", "David", "Beatriz"]

-- 1. Transformaciones con map
cuads = map (^2) nums

inits = map head noms

-- 2. Filtrado con filter
pars = filter even nums

nomsLargos = filter (\x -> length x > 5) noms

-- 3. Combinaciones
cuadsPars = map (^2) (filter even nums)
```

Tareas:

1. Crea una función que devuelva los cubos de los números impares
2. Filtra los nombres que contienen la letra 'a' y conviértelos a mayúsculas
3. Implementa una función que sume 10 a los números mayores que 5

Ejercicio 3: Functores con Listas Anidadas (20 min)

Trabaja con estructuras de datos anidadas y functores.

```
-- Listas anidadas
listAnid = [[1, 2, 3], [4, 5], [6, 7, 8, 9]]

-- Transformaci n con doble fmap
dupAnid = fmap (fmap (*2)) listAnid

-- Aplanar lista
aplana = concat

-- Functor para procesamiento en profundidad
procProfundo = fmap (fmap show)
```

Tareas:

1. Crea una función que calcule la suma de cada sublista
2. Implementa una función que filtre las sublistas con más de 2 elementos
3. Crea una transformación que convierta todos los números a string con prefijo "N-"

Ejercicio 4: Aplicaciones Prácticas con Datos Reales (25 min)

Aplica functores y listas a un escenario del mundo real.

```haskell
-- Tipo para representar productos
data Producto = Producto {
    nom :: String,
    pre :: Double,
    cant :: Int
} deriving Show

-- Datos de ejemplo
inv = [
    Producto "Laptop" 1200.50 5,
    Producto "Mouse" 25.99 20,
    Producto "Teclado" 75.30 12,
    Producto "Monitor" 350.00 8
]

-- Funci n auxiliar para valor
valorProd p = pre p * fromIntegral (cant p)

-- 1. Obtener todos los nombres
nomsProds = map nom inv

-- 2. Filtrar productos caros
prodsCaros = filter (\p -> pre p > 100.0) inv

-- 3. Calcular valor total por producto
vInventario = map valorProd inv

-- 4. Aplicar descuento del 10%
aplDesc = map (\p -> p {pre = pre p * 0.9})
```

Tareas:

1. Calcula el valor total del inventario
2. Filtra los productos con menos de 10 unidades
3. Crea una lista de strings con formato "Nombre: Precio"
4. Implementa un descuento escalonado

Ejercicios Adicionales

```haskell
-- 1. Functor para tipos personalizados complejos
data Arbol a = Hoja a | Nodo (Arbol a) (Arbol a) deriving Show

instance Functor Arbol where
    fmap f (Hoja x) = Hoja (f x)
    fmap f (Nodo izq der) = Nodo (fmap f izq) (fmap f der)
```

```haskell
-- 2. Procesamiento de texto
procTexto = fmap (filter (/= ' ') . map toUpper)

-- 3. Calculadora estad stica
stats xs = (prom, min, max)
    where
        prom = sum xs / fromIntegral (length xs)
        min = minimum xs
        max = maximum xs
```

Reto Final

```haskell
-- Sistema de procesamiento de pedidos
data Pedido = Pedido {
    cli :: String,
    items :: [String],
    tot :: Double,
    desc :: Bool
} deriving Show

peds = [
    Pedido "Ana" ["Laptop", "Mouse"] 1226.49 True,
    Pedido "Carlos" ["Teclado"] 75.30 False,
    Pedido "Elena" ["Monitor", "Teclado"] 425.30 True
]

-- Tareas del reto:
-- 1. Aplicar 15% de descuento a pedidos con descuento
-- 2. Filtrar pedidos con total mayor a 500.00
-- 3. Crear resumen por cliente
-- 4. Calcular el total general de todos los pedidos
```

Evaluación

- Correcta implementación de instancias de Functor
- Uso apropiado de map, filter y funciones de orden superior
- Manejo adecuado de listas anidadas
- Soluciones eficientes y elegantes
- Manejo de casos bordes y validaciones

Recursos Adicionales

- Documentación oficial de Haskell: https://www.haskell.org/
- Hoogle para búsqueda de funciones: https://hoogle.haskell.org/
- Learn You a Haskell: http://learnyouahaskell.com/