Guía de Laboratorio: Tipos de Datos y Control de Programas en Haskell

Programación Funcional

Duración: 1.5 horas

Objetivos de la Práctica

- Comprender y utilizar tipos de datos algebraicos en Haskell
- Implementar funciones con pattern matching
- Manejar el control de programas usando corte y fallo
- Desarrollar habilidades en la manipulación de listas y tipos personalizados

1 Introducción Teórica

1.1 Tipos de Datos Algebraicos

Los tipos de datos algebraicos permiten definir estructuras de datos complejas mediante:

- **Product types**: Tuplas y registros
- **Sum types**: Tipos con múltiples constructores
- **Recursive types**: Estructuras que se referencian a sí mismas

1.2 Funciones de Control: Corte y Fallo

- **Corte (Short-circuit evaluation)**: Evaluación perezosa que evita cálculos innecesarios
- **Fallo**: Manejo de casos donde las operaciones pueden no tener resultado definido

2 Ejercicios Prácticos

2.1 Ejercicio 1: Tipos de Datos Básicos (15 minutos)

Define los siguientes tipos de datos y funciones:

```haskell
-- 1. Tipo para representar d as de la semana
data Dia = Lunes | Martes | Miercoles | Jueves | Viernes | Sabado |
   Domingo

-- 2. Funci n que determina si un d a es laborable
esLaborable :: Dia -> Bool
esLaborable Sabado = False
esLaborable Domingo = False
esLaborable _ = True

-- 3. Tipo para representar coordenadas 2D
data Punto = Punto Double Double

-- 4. Funci n que calcula la distancia entre dos puntos
distancia :: Punto -> Punto -> Double
distancia (Punto x1 y1) (Punto x2 y2) = sqrt ((x2 - x1)^2 + (y2 -
   y1)^2)
```

Tarea: Implementa una función que calcule el punto medio entre dos puntos.

2.2 Ejercicio 2: Tipos Algebraicos (20 minutos)

Crea tipos más complejos y funciones asociadas:

```haskell
-- 1. Tipo para formas geom tricas
data Forma = Circulo Double          -- radio
           | Rectangulo Double Double -- base, altura
           | Cuadrado Double         -- lado
           deriving Show

-- 2. Funci n para calcular  rea
area :: Forma -> Double
area (Circulo r) = pi * r * r
area (Rectangulo b a) = b * a
area (Cuadrado l) = l * l

-- 3. Tipo para  rbol  binario
data Arbol a = Hoja a
             | Nodo (Arbol a) a (Arbol a)
             deriving Show

-- 4. Funci n para calcular altura del  rbol
altura :: Arbol a -> Int
altura (Hoja _) = 1
altura (Nodo izq _ der) = 1 + max (altura izq) (altura der)
```

Tarea: Implementa una función que cuente el número de hojas en un árbol.

2.3 Ejercicio 3: Control con Corte (20 minutos)

Implementa funciones que utilicen evaluación perezosa:

```haskell
-- 1. Funci n que encuentra el primer elemento que cumple una
   condici n
encontrar :: (a -> Bool) -> [a] -> Maybe a
encontrar _ [] = Nothing
encontrar f (x:xs)
    | f x       = Just x
    | otherwise = encontrar f xs

-- 2. Funci n que verifica si todos los elementos cumplen una
   condici n
todos :: (a -> Bool) -> [a] -> Bool
todos _ [] = True
todos f (x:xs) = f x && todos f xs

-- 3. Funci n que toma elementos mientras cumplan una condici n
tomarMientras :: (a -> Bool) -> [a] -> [a]
tomarMientras _ [] = []
tomarMientras f (x:xs)
    | f x       = x : tomarMientras f xs
    | otherwise = []
```

Tarea: Crea una función que concatene dos listas solo si la primera no está vacía.

2.4 Ejercicio 4: Manejo de Fallo (20 minutos)

Implementa funciones que manejen casos de fallo:

```haskell
-- 1. Divisi n segura
divisionSegura :: Double -> Double -> Maybe Double
divisionSegura _ 0 = Nothing
divisionSegura x y = Just (x / y)

-- 2. Acceso seguro a lista
elementoEn :: Int -> [a] -> Maybe a
elementoEn _ [] = Nothing
elementoEn 0 (x:_) = Just x
elementoEn n (_:xs) = elementoEn (n-1) xs

-- 3. Tipo para resultados con error
data Resultado a = Exito a | Error String

-- 4. Funci n que procesa una lista de operaciones
procesarLista :: [a -> Maybe b] -> [a] -> [Maybe b]
procesarLista ops valores = zipWith (\f x -> f x) ops valores
```

Tarea: Implementa una función que convierta `Maybe a` a `Resultado a`.

2.5 Ejercicio 5: Integración (15 minutos)

Combina todos los conceptos aprendidos:

```haskell
-- 1. Sistema de gesti n de estudiantes
data Estudiante = Estudiante {
    nombre :: String,
    edad :: Int,
    calificaciones :: [Double]
} deriving Show

-- 2. Funci n que calcula el promedio con manejo de error
promedio :: Estudiante -> Maybe Double
promedio (Estudiante _ _ []) = Nothing
promedio (Estudiante _ _ notas) =
    Just (sum notas / fromIntegral (length notas))

-- 3. Funci n que filtra estudiantes aprobados
aprobados :: [Estudiante] -> [Estudiante]
aprobados = filter (\e -> case promedio e of
                          Just p -> p >= 6.0
                          Nothing -> False)
```

Tarea: Crea una función que encuentre al estudiante con el promedio más alto.

Evaluación

- Correcta definición de tipos de datos (25 %)
- Implementación adecuada de funciones (35 %)
- Manejo apropiado de casos de fallo (20 %)
- Uso eficiente de evaluación perezosa (20 %)

Material Adicional

- **Documentación**: Haskell Language Report
- **Libros**: "Learn You a Haskell for Great Good!"
- **Recursos online**: Hoogle, Haskell Wiki