Curso: Programación Lógica y Funcional

Unidad 1: Programación funcional

Sesión 6: Funciones de Control - Corte y Fallo

Docente: Carlos R. P. Tovar

Universidad Tecnológica del Perú

Dudas de la anterior sesión

INICIO
Objetivo de la sesión

- Comprender los conceptos de terminación temprana (corte)
- Implementar manejo de fallos controlados
- Utilizar técnicas de evaluación perezosa para control de ejecución
- Resolver problemas con diferentes estrategias de control

TRANSFORMACIÓN
Terminación Temprana (Corte)

```
-- Corte con guards y pattern matching
factorial :: Int -> Int
factorial n
    | n < 0     = error "Corte: número negativo"  -- Corte por error
    | n == 0    = 1                               -- Corte exitoso
    | otherwise = n * factorial (n-1)
```

```
-- Corte con case expressions
validarEdad :: Int -> String
validarEdad edad = case edad of
    n | n < 0 -> "Corte: edad negativa"
    0 -> "Corte: recién nacido"
    _ -> "Edad válida"
```

Manejo de Fallos Controlados

```
-- Maybe para fallos controlados
buscarElemento :: Eq a => a -> [a] -> Maybe a
buscarElemento _ [] = Nothing  -- Fallo controlado
buscarElemento x (y:ys)
   | x == y    = Just x      -- Corte exitoso
   | otherwise = buscarElemento x ys

-- Either para fallos con información
dividirConMensaje :: Float -> Float -> Either String Float
dividirConMensaje _ 0 = Left "Fallo: división por cero"  -- Corte con fallo
dividirConMensaje x y = Right (x / y)              -- Éxito
```

Ejercicio 1: Búsqueda con Corte

```
-- Buscar primer elemento que cumpla condición
buscarPrimero :: (a -> Bool) -> [a] -> Maybe a
buscarPrimero _ [] = Nothing
buscarPrimero f (x:xs)
   | f x      = Just x  -- Corte al encontrar el primero
   | otherwise = buscarPrimero f xs

-- Ejemplo:
-- buscarPrimero (>5) [1, 3, 7, 2, 9] → Just 7
```

Ejercicio 2: Validación con Múltiples Cortes

```
validarContraseña :: String -> Either String String
validarContraseña pwd
   | length pwd < 8 = Left "Muy corta"
   | not (any isUpper pwd) = Left "Sin mayúsculas"
   | not (any isLower pwd) = Left "Sin minúsculas"
   | not (any isDigit pwd) = Left "Sin números"
   | otherwise = Right pwd  -- Todas las validaciones pasaron

-- Cada guard actúa como punto de corte potencial
```

Evaluación Perezosa como Corte Natural

```
-- La evaluación perezosa provee corte automático
cortePerezoso :: Bool -> Int -> Int
cortePerezoso True x = x * 2
cortePerezoso False _ = error "Corte: condición falsa"

-- El segundo argumento no se evalúa si el primero es False
resultado = cortePerezoso False (product [1..1000000])
-- El cálculo grande nunca se ejecuta
```

Short-Circuit Evaluation

```
-- && y || tienen corte automático
evaluacionCortocircuito :: Bool -> Bool -> Bool
evaluacionCortocircuito a b = a && b

-- Si a es False, b nunca se evalúa
ejemplo = False && (error "No se evalúa este error")

-- Implementación personalizada
(&&|) :: Bool -> Bool -> Bool
True &&| True = True
True &&| False = False
False &&| _ = False  -- Corte: segundo argumento no evaluado
```

Ejercicio 3: Sistema de Permisos con Corte

```
data Usuario = Usuario {
    nombre :: String,
    edad :: Int,
    tienePermiso :: Bool
}

accesoSistema :: Usuario -> Either String String
accesoSistema usuario
    | edad usuario < 18 = Left "Corte: menor de edad"
    | not (tienePermiso usuario) = Left "Corte: sin permisos"
    | otherwise = Right ("Bienvenido " ++ nombre usuario)

-- Múltiples puntos de corte potenciales
```

Manejo de Fallos en Cadenas de Procesamiento

```
-- Cadena de procesamiento con posibles fallos
procesarDatos :: String -> Either String Int
procesarDatos input = do
    paso1 <- parsearEntero input  -- Puede fallar
    paso2 <- validarPositivo paso1  -- Puede fallar
    paso3 <- aplicarTransformacion paso2  -- Puede fallar
    return paso3

-- Cada paso representa un posible punto de corte
```

Búsqueda en Árbol con Corte

```
data Arbol a = Vacio | Nodo a (Arbol a) (Arbol a)

-- Buscar valor en árbol con corte al encontrarlo
buscarEnArbol :: Eq a => a -> Arbol a -> Bool
buscarEnArbol _ Vacio = False
buscarEnArbol x (Nodo valor izq der)
   | x == valor = True  -- Corte: encontrado
   | otherwise = buscarEnArbol x izq || buscarEnArbol x der
   -- || provee corte si izq devuelve True
```

Patrones de Corte con where/let

```
procesamientoComplejo :: Int -> Maybe (Int, Int)
procesamientoComplejo x
   | x < 0 = Nothing  -- Corte temprano
   | otherwise = Just (doble, triple)
   where
      doble = x * 2
      triple = x * 3
      -- Cálculos solo se ejecutan si x >= 0
```

Ejercicio 5: Validación Anidada con Corte

```
validarFormulario :: String -> Int -> Either String (String, Int)
validarFormulario nombre edad = do
  nombreValido <- if null nombre
                  then Left "Corte: nombre vacío"
                  else Right nombre
  edadValida <- if edad < 0
                then Left "Corte: edad negativa"
                else Right edad
  return (nombreValido, edadValida)

-- Múltiples puntos de corte potenciales
```

PRACTICA
Tarea para la Próxima Sesión

- Implementar un sistema de validación con 5 niveles de corte
- Crear una función de búsqueda que corte al encontrar el resultado
- Desarrollar un procesador de datos con manejo de fallos en cadena
- Implementar un árbol de decisiones con puntos de corte múltiples

Recursos:

- Haskell Error Handling
- Learn You a Haskell - Input and Output

CIERRE
Conclusiones

- ¿Cómo implementarías un sistema de corte en una función recursiva?
- ¿Qué ventajas tiene el uso de Maybe/Either sobre excepciones?
- ¿Cómo afecta la evaluación perezosa a las estrategias de corte?
- ¿Cuándo es apropiado usar error vs Maybe/Either?

Cierre

-- En Haskell, el "corte" se maneja mediante
-- pattern matching, guards y tipos algebraicos
-- proporcionando control de flujo seguro
-- y predecible

¡El control en Haskell es declarativo y basado en tipos! 🚀

Universidad
Tecnológica
del Perú