Curso: Programación Lógica y Funcional

Unidad 2: Programación funcional

Sesión 6: Funciones de tipo recursivo

Docente: Carlos R. P. Tovar

UTP Universidad Tecnológica del Perú

INICIO
Objetivo de la sesión

- Al finalizar la sesión, el estudiante será capaz de **definir y aplicar funciones recursivas en Haskell para resolver problemas de programación**, utilizando *casos base* y *llamadas recursivas* de forma eficiente.

Dudas de la sesión anterior

- ¿Qué son las funciones de control (corte y fallo) en Haskell?
- ¿En qué casos prácticos se utilizan?
- ¿Qué dificultades tuvieron en los ejercicios autónomos?

UTILIDAD

- La recursión reemplaza a los bucles en Haskell.
- Permite resolver problemas como:
  - Cálculo de factoriales
  - Procesamiento de listas.
  - Definición de secuencias matemáticas (ej. Fibonacci).
- Es un pilar de la programación funcional y de la resolución de problemas declarativos.

TRANSFORMACIÓN
Definición de recursividad

- Un problema se resuelve en términos de una versión más pequeña de sí mismo.
- Toda función recursiva debe tener:
  - Caso base → punto de parada.
  - Caso recursivo → llamada a sí misma.

- Ejemplo 1: Factorial

```
factorial :: Int -> Int
factorial 0 = 1
factorial n = n * factorial (n-1)
```

Definición de recursividad: Fibonacci

```
fibo :: Int -> Int
fibo 0 = 0
fibo 1 = 1
fibo n = fibo (n-1) + fibo (n-2)
```

factorial 4 → 4 * factorial 3

factorial 3 → 3 * factorial 2

factorial 2 → 2 * factorial 1

factorial 1 → 1 * factorial 0

factorial 0 = 1

Ejercicios guiados en clase

1. Definir una función recursiva que calcule la suma de los primeros n números.
2. Implementar una función que cuente los elementos de una lista.
3. Crear una función que invierta una lista.

PRACTICA
Ejercicios para resolver en clase/tarea

1. Implementa una función recursiva potencia(base, exp) que calcule la potencia de un número.
2. Define una función que determine el máximo de una lista.
3. Implementa una versión optimizada de Fibonacci usando acumuladores (tail recursion).
4. Crea una función que determine si una lista es palíndroma.

CIERRE
Conclusiones

- La recursividad es el mecanismo fundamental para repetir procesos en Haskell.
- Siempre debe existir un caso base para evitar recursión infinita.
- Se aplica en problemas matemáticos y manipulación de estructuras de datos.
- Desarrollar funciones recursivas mejora la lógica algorítmica y el pensamiento declarativo.

Reflexión final

¿Qué ventajas ofrece la recursividad frente a los bucles tradicionales?

¿En qué escenarios sería mejor evitar la recursión?

Universidad
Tecnológica
del Perú