

# Universidad Tecnológica Nacional Facultad Regional Resistencia

### Paradigmas de Programación

Programación Funcional



### Modelo Funcional



### Características:



- Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones puramente matemáticas.
- **Brevedad**. Los programas son más cortos y concisos.
- No existen las asignaciones de variables ni construcciones estructuradas como la secuencia o la iteración.

(se llevan a cabo por medio de funciones recursivas).

- En el esquema funcional se evalúan las expresiones mediante un proceso de reducción.
- Transparencia referencial. El valor que devuelve una función está únicamente determinado por el valor de sus argumentos.
- Funciones de orden superior. Las funciones pueden recibir como argumento otras y funciones y también devolverlas como resultado.



# Reducción de Expresiones



 La labor de un evaluador es calcular el resultado que se obtiene al simplificar una expresión utilizando las definiciones de las funciones involucradas.

```
Ej: doble :: Integer \rightarrow Integer doble x = x + x

5 * doble 3
5 * (3 + 3)
5 * 6
5 * 6
6 por el operador * }

30
```

Un **redex** es cada parte de la expresión que pueda reducirse. Puede ser **de afuera hacia adentro** o **de adentro hacia fuera**.

Cuando una expresión no puede ser reducida mas se dice que esta en **forma normal**.



# Reducción de Expresiones Estrategia de Control



- Regla de Selección o Cálculo: para determinar el redex a reducir.
  - Evaluación Impaciente
  - Evaluación Perezosa

Regla de Búsqueda: para determinar la ecuación a utilizar. Es igual para todos los sistemas de programación funcional.



### Ordenes de reducción

### **Orden Impaciente**



- Se reduce siempre el término MAS INTERNO (el más anidado en la expresión).
- En caso de que existan varios términos a reducir (con la misma profundidad) se selecciona el que aparece más a la izquierda de la expresión.
- Esto también se llama paso de parámetros por valor (call by value)

```
doble (doble 3)
doble (3 + 3)
doble (6)
6 + 6
12
```

por la definición de doble por el operador + por la definición de doble por el operador +



doble :: Integer  $\rightarrow$  Integer doble x = x + x

### Ordenes de reducción Orden Normal



- Consiste en seleccionar el término MÁS EXTERNO (el menos anidado), y en caso de conflicto el que aparezca más a la izquierda de la expresión.
- Esta estrategia se conoce como "paso de parámetro por nombre o referencia" (call by name), ya que se pasan como parámetros de las funciones expresiones en vez de valores.
- A los evaluadores que utilizan el orden normal se les llama "no estrictos".

```
doble (doble 3)
(doble 3) + (doble 3)
(3 + 3) + (doble 3)
6 + (doble 3)
6 + (3 + 3)
6 + 6
12
```

por la definición de doble por la definición de doble por la definición de + por la definición de doble por la definición de + por la definición de +



doble :: Integer  $\rightarrow$  Integer doble x = x + x

### Ordenes de reducción

### Evaluación Perezosa (Lazy)



- No se evalúa ningún elemento en ninguna función hasta que no sea necesario.
- La evaluación perezosa consiste en utilizar paso por nombre y recordar los valores de los argumentos ya calculados para evitar recalcularlos.
- También se denomina estrategia de pasos de parámetros por necesidad (call by need).

```
doble (doble 3)por la definición de doblea + a donde a = doble 3por la definición de doblea + a donde a = b + b donde b = 3por el operador +a + a donde a = 6por el operador +12
```



### Evaluación Perezosa I

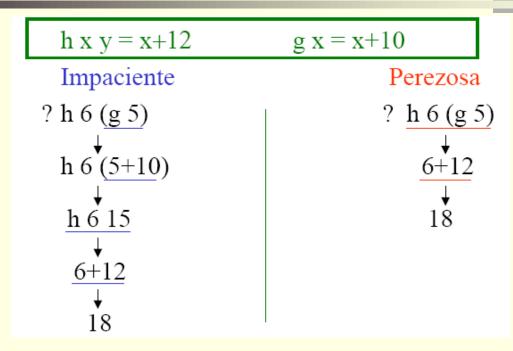


$f \times y = x*2+y$	$g_{X} = x+10$
Impaciente	Perezosa
? f 3 (g 5)	? <u>f</u> 3 (g 5)
f 3 ( <u>5+10</u> )	3*2+(g 5)
f 3 15	<b>↓</b>
<b>↓</b>	$ \begin{array}{c} 6 + (\underline{g} 5) \\ \downarrow \end{array} $
<u>3*2</u> + 15	6 + (5+10)
6+15	6+15
21	21



### Evaluación Perezosa II



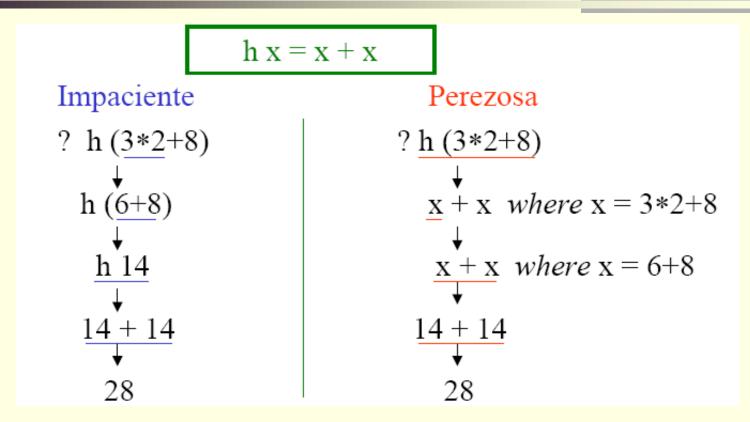


- Funciones no Estrictas: los argumentos se evalúan solo si es necesario.
  - f x y = y + 10
  - floop 3
  - f (fib 31515) 4



### Evaluación Perezosa III





Los argumentos se evalúan como máximo una vez.



# Transparencia Referencial



- Si una función es llamada dos veces con los mismos parámetros, se obtendrá siempre el mismo resultado. El resultado depende única y exclusivamente de los parámetros provistos y no produce efecto de lado.
  - Decimos que una operación tiene transparencia referencial si es:
    - Independiente: No dependen del estado de nada que este fuera de si misma
    - Sin estado/Stateless: No tiene un estado que se mantenga de llamada en llamada
    - Determinística: Siempre devuelven el mismo valor dados los mismos argumentos

### Efecto de Lado/Colateral (Side Effect)

- Hay efecto de lado cuando un cambio de estado sobrevive a la realización de una operación.
- Ej: Modificación de una variable global; uso de variable estáticas; fecha y hora del sistema; valores ingresados por el usuario; valores aleatorios.



# Funciones de orden superior I



Mapea los valores de una lista con una función dada como parámetro.

Un lenguaje utiliza funciones de orden superior cuando permite que las funciones sean tratadas como valores de 1ª clase, permitiendo que sean pasadas como argumentos de funciones y que sean devueltas como resultados.

```
map :: (t -> u) -> [t] -> [u]
map f [] = []
map f (x:xs) = f x : map f xs
```

```
map cuad [1,2,3,4] = [1, 4, 9, 16]

map impar [1,2,3,4] = [True, False, True, False]

map (*) [1..5] = [(1*), (2*), (3*), (4*), (5*)]

map (suma 3) [1,2,3,4] = [4, 5, 6, 7]
```

En el último ejemplo, la función que se pasa está parcialmente instanciada; o sea: 3 + y.



## Funciones de orden superior II



### Ejemplo 2

```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith _ [] _ = []
zipWith _ _ [] = []
zipWith f (x:xs) (y:ys) = f x y : zipWith f xs ys

flip :: (a -> b -> c) -> (b -> a -> c)
flip f = g
    where g x y = f y x
```

Toma una función y dos listas y las une aplicando la función entre los correspondientes parámetros

Devuelve una función similar a la original con los dos parámetros intercambiados

```
zipWith (+) [2..] [2..10]
      [4,6,8,10,12,14,16,18,20]

zipWith (++) ["Bart ", "Lisa ", "Maggie "] (replicate 3 "Simpson")
      ["Bart Simpson","Lisa Simpson","Maggie Simpson"]

zipWith (flip (++)) (replicate 3 "Simpson") ["Bart ", "Lisa ", "Maggie "]
      ["Bart Simpson","Lisa Simpson","Maggie Simpson"]
```



# Sistemas de Inferencia de Tipos 🔀



- Muchos lenguajes funcionales han adoptado un sistema de inferencia de tipos que consiste en:
  - El programador no está obligado a declarar el tipo de las expresiones.
  - El compilador contiene un algoritmo que infiere el tipo de las expresiones.
  - Si el programador declara el tipo de alguna expresión, el sistema chequea que el tipo declarado coincide con el tipo inferido.
  - Los sistemas de inferencia de tipos aumentan su flexibilidad mediante la utilización de polimorfismo.



### Polimorfismo



### Polimorfismo Paramétrico:

Una función puede recibir parámetros de diferentes tipos; operará uniformemente sobe cualquiera de ellos.

# Ejemplos long :: [a] -> Int long [] = 0 long (x:xs) = 1 + long xs toma :: Num a => a -> [b] -> [b] toma 0 x = [] toma \_ [] = [] toma n (x:xs) = x:toma (n-1) xs



### Haskell



Introducción al Lenguaje Haskell



# Elementos de la Prog. Funcional Expresiones Básicas



- Expresiones de tipo: formadas con las constructoras de tipo
  - (t1,t2)
- t1|t2
- t1->t2
- Asignaciones de tipo
  - a::t
- Expresiones funcionales: controladas por las asignaciones de tipo
  - Aplicación de una función: a+a f a
  - Abstracción de una función a partir de una expresión: λa.a+a
- Asignaciones nombre
  - A expresiones de tipo
    - typet = (t1,t2)
  - A expresiones funcionales
    - g = a + a



### Estructuras de datos infinitas



- $\blacksquare$  countFrom n = n : countFrom <math>(n+1)
  - Construye un alista infinita con todos los números mayores o iguales que n.
- countFrom 1
  - **1** [1, 2, 3, 4, 5, 6, 7, 8, 9, ...]
- take 10 (countFrom 1)
  - Toma los 10 primeros elementos de una lista.
- sum (take 10 (countFrom 1))
  - Suma los elementos de la lista.
  - Resultado: 55



# Estructuras de datos infinitas II



Definiciones por comprensión: significa definir una función dando una característica general de cómo se construyen los elementos de la dominio y la imagen.







