

Paradigmas de Programação

# Linguagem Haskell

Profª Andréa Schwertner Charão  
DLSC/CT/UFSM

# Gerando listas

- Notação '..'
- Listas infinitas
- Listas por compreensão

# Listas: ellipsis notation ('..')

- Notação usada para **gerar** listas que representam domínios ordenados (números, caracteres, etc.)
- Exemplo:

> [1..5]

[1,2,3,4,5]

## Listas: ellipsis notation ('..')

- Notação **[n..m]**: lista de n a m, com passo 1 (m é o limite superior)
- Notação **[n,m..p]**: lista de n a p, com passo m-n (passo pode ser negativo)
- Notação **[n..]**: lista infinita, com passo 1
- Notação **[n,m..]**: lista infinita, com passo m-n

# Listas: notação [n..m]

> [1..5]

[1,2,3,4,5]

> [5..1]

[]

> [1.66 .. 5] -- 5 é o limite

[1.66,2.66,3.66,4.66]

> ['a'..'z']

"abcdefghijklmnopqrstuvwxyz"

# Listas: notação [n,m .. p]

> [2,4..10] -- passo  $4-2 = 2$

[2,4,6,8,10]

> [3,6..16] -- passo  $6-3 = 3$

[3,6,9,12,15]

> [10, 9 .. 1] -- passo  $9-10 = -1$

[10,9,8,7,6,5,4,3,2,1]

> [5, 4.5 .. 1] -- passo  $4.5-5 = -0.5$

[5.0,4.5,4.0,3.5,3.0,2.5,2.0,1.5,1.0]

> ['a','c'.. 'z']

"acegikmoqsuwy"

# Geração de listas com notação '..'

- Pode usar valores literais ou nomes de argumentos, expressões, etc.
- Atenção aos tipos! (usar conversão se necessário)

```
func :: Int -> Float -> [Float]
func c x = [0.0, x..x*fromIntegral (c-1)]
```

# Listas: notação [n..]

- Lista **infinita**, com passo 1

```
> [10..]
```

```
[10,11,12,13 ^C Interrupted.
```

```
> [0.33..]
```

```
[0.33,1.33,2.33 ^C Interrupted.
```



# Listas: notação [n,m..]

- Lista **infinita**, com passo m-n

> [0.45, 0.75 ..]

[0.45,0.75,1.05,1.35 ^C Interrupted.

# Avaliação preguiçosa (*lazy evaluation*)

- Retarda a avaliação de uma expressão até que seu valor seja realmente necessário
- Sinônimos: call-by-need, non-strict evaluation
- Exemplo:

```
> take 5 [11..]
```

```
[11,12,13,14,15]
```

```
> take 5 (cycle [1,2,3])
```

```
[1,2,3,1,2]
```

# Avaliação preguiçosa (*lazy evaluation*)

- Vantagem no desempenho, pois evitam-se cálculos desnecessários
- Vantagem no uso de memória, pois valores são criados somente quando necessário
- Vantagem no poder de expressão da linguagem

# Listas por compreensão

- Do inglês "*list comprehension*"
- Recurso em Haskell inspirado numa notação de conjuntos em matemática
- Conjuntos: "por extensão"X "por compreensão"
  - Por extensão: enumera-se os elementos
  - Por compreensão: define-se uma regra de geração dos elementos

# Listas por compreensão

## ■ Exemplo em matemática

- $S = \{ 2.x \mid x \in \mathbb{N}, x \leq 10 \}$
- S é o conjunto dos números naturais menores ou iguais a 10, multiplicados por 2
- Ou ainda: S é o conjunto dos 10 primeiros números pares

## ■ Em Haskell: `[x*2 | x <- [0..9]]`

Resultado: `[0,2,4,6,8,10,12,14,16,18]`

# Listas por compreensão

## ■ Forma geral:

[ exprsaida | gerador\_1, ..., gerador\_n]

## ■ Onde:

- **gerador** \* tem a forma "padrao <- expressao" e representa uma lista de origem
- **exprsaida** é uma expressão avaliada sobre as listas geradas do lado direito
- o símbolo "<-" pode ser lido como "pertence"

## ■ Exemplo:

```
> [x*2 | x <- [0..9]]  
[0,2,4,6,8,10,12,14,16,18]
```

# Listas por compreensão

## ■ Outro exemplo:

```
> [(x,y) | x <- [1,2], y <- [1,2]]  
[(1,1),(1,2),(2,1),(2,2)]
```

## ■ Onde:

- x e y são as listas de origem
- listas x e y são combinadas elemento por elemento
- saída são tuplas (x,y)

# Listas por compreensão

- Outro exemplo:

```
> [(x,y) | x <- [1..3], y <- ['a','b']]  
[(1,'a'),(1,'b'),(2,'a'),(2,'b'),(3,'a'),(3,'b')]
```



# Listas por compreensão

- Filtro para strings (.hs)

```
removeChar :: Char -> [Char] -> [Char]
removeChar c str = [x | x <- str, x /= c]
```

Lista geradora é reduzida  
com esta condição

- Exemplo de uso:

```
> removeChar 'a' "abababa"
"bbb"
```

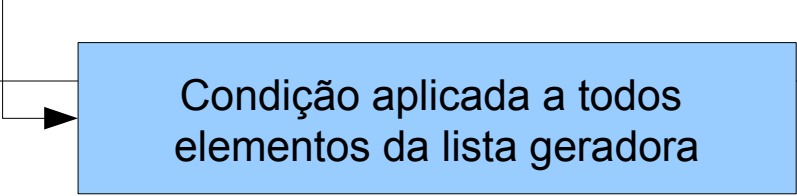
# Listas por compreensão

## ■ Filtro para imagens (.hs)

```
filterImg :: [Int] -> [Int]
```

```
filterImg bitmap =
```

```
  [if pixel < 10 then 0 else pixel | pixel <- bitmap]
```



Condição aplicada a todos  
elementos da lista geradora

## ■ Exemplo de uso:

```
> filterImg [1,3,4,20,40,40,40,3,2]
```

```
[0,0,0,20,40,40,40,0,0]
```