

Generalizações

Márcio Lopes Cornélio

(a partir de *slides* elaborados por André Santos, Sérgio Soares e Fernando Castor)

Funções de alta ordem

- Recebem outras funções como **argumentos** ou as produzem como **resultado**.
- Permitem
 - definições polimórficas
 - funções aplicadas sobre uma coleção de tipos
 - padrões de recursão usados por várias funções.

Exemplos

```
applyTwice :: (a -> a) -> a -> a
applyTwice f x = f (f x)
```

- Funções tomam um argumento e retornam uma função parcialmente aplicada

Por que funções de alta ordem

- Facilitam **entendimento** das funções
- Facilitam **modificações** (mudança na função de transformação)
- Aumentam **reuso** de definições/código
 - modularidade
 - Ex. usar a função `map` para remarcar o valor de uma lista de preços

Exemplo: análise de vendas

```
total :: (Int->Int)-> Int -> Int
total f 0 = f 0
total f n = total f (n-1) + f n

totalVendas n = total vendas n

sumSquares :: Int -> Int
sumSquares n = total sq n
```

Outros exemplos

```
maxFun :: (Int -> Int) -> Int -> Int
maxFun f 0 = f 0
maxFun f n = maxi (maxFun f (n-1)) (f n)

zeroInRange :: (Int -> Int) -> Int -> Bool
zeroInRange f 0 = (f 0 == 0)
zeroInRange f n = zeroInRange f (n-1)
                  || (f n == 0)
```

Exercício

- Dada uma função, verificar se ela é crescente em um intervalo de 0 a n

```
isCrescent :: (Int -> Int) -> Int -> Bool
```

Exemplos

```
double :: [Int] -> [Int]
double [] = []
double (a:x) = (2*a) : double x

sqrList :: [Int] -> [Int]
sqrList [] = []
sqrList (a:x) = (a*a) : sqrList x
```

Funções de **mapeamento** (*mapping*)

Exemplos

```
times2 :: Int -> Int
times2 n = 2 * n
```

```
sqr :: Int -> Int
sqr n = n * n
```

Funções de **transformação** dos elementos

A função de mapeamento

- Recebe como argumentos
 - a **transformação** a ser aplicada a cada elemento da lista
 - uma função
 - a lista de entrada

map

```
map :: (t -> u) -> [t] -> [u]
map f [] = []
map f (a:as) = f a : map f as

doubleList xs = map times2 xs
sqrList xs = map sqr xs

snds :: [(t,u)] -> [u]
snds xs = map snd xs

map length ["abc", "defg"] = ?
```

Outra definição para map

```
map f l = [f a | a <- l]
```

Exemplo: *folding*

```
sumList :: [Int] -> Int
sumList [] = 0
sumList a:as = a + sumList as
e1 + e2 + ... + em

fold :: (t -> t -> t) -> [t] -> t
fold f [a] = a
fold f (a:as) = f a (fold f as)

sumList l = fold (+) 1
```

fold=foldr1 no ghci

Exemplo: *folding*

```
and :: [Bool] -> Bool
and xs = fold (&&) xs

concat :: [[t]] -> [t]
concat xs = fold (++) xs

maximum :: [Int] -> Int
maximum xs = fold maxi xs
```

Exemplo: *folding*

```
fold (||) [False, True, True]
fold (++) ["Bom", " ", "Dia"]
fold min [6]
fold (*) [1..6]
```

foldr

```
foldr :: (t -> u -> u) -> u -> [t] -> u
foldr f s [] = s
foldr f s (a:as)
    = f a (foldr f s as)

concat :: [[t]] -> [t]
concat xs = foldr (++) [] xs

and :: [Bool] -> Bool
and bs = foldr (&&) True bs
```

fold=foldr1 no ghci

Exemplo: filtrando

```
digits, letters :: String -> String
filter :: (t -> Bool) -> [t] -> [t]
filter p [] = []
filter p (a:as)
    | p a = a : filter p as
    | otherwise = filter p as

digits st = filter isDigit st
letters st = filter isLetter st

evens xs = filter isEven xs
    where isEven n = (n `mod` 2 == 0)
```

Outra definição para filter

```
filter p l = [a | a <- l, p a]
```

Exercícios

- Defina as seguintes funções sobre listas
 - eleva os itens ao quadrado
 - *mapping*
 - retorna a soma dos quadrados dos itens
 - *folding*
 - manter na lista todos os itens maiores que zero.
 - *filtering*

Exemplo: Biblioteca

```
livros :: BancoDados -> Pessoa -> [Livro]
livros bd pes = map snd (filter isPess bd)
where isPess (p,l) = (p == pes)
```

```
devolver :: BancoDados -> Pessoa
           -> Livro -> BancoDados
devolver bd p l = filter notPL bd
where notPL t = (t /= (p,l))
```

Exercícios

- **O que a função**
`naosei l = foldr (++) [] (map sing l)`
, onde `sing a = [a]`, faz?
- **Defina uma função**
`maiores :: [[Int]] -> [Int]`
que, dada uma lista de listas de inteiros, devolve uma lista contendo o maior elemento de cada sub-lista da entrada