Linguagem de Programação
Introdução à Linguagem Hugs98

TURMA A
PROF. MARCELO LADEIRA CIC/UNB

Funções de Ordem alta

Funções que manuseiam outras funções:

- composição de funções.
- passadas como parâmetros.
- resultado da chamada de função.
- aplicação parcialmente aos seus argumentos.
- usadas em estrutura de dados.

Composição de Funções

```
f . g x = f (g x)
var xs = (sum . map (^2)) xs / n - (sum xs / n)^2
      where n = length xs
Main> var [1.0,1.5..5.0]
  1.66667 :: Double
Main> ((^3) . (/2)) 10
  125.0 :: Double
Main> (sum . take 10) [1..]
  55 :: Integer
Main> let uns = 1:uns in (sum . take 20) uns
  20 :: Integer
Main> take 20 uns where uns = 1:uns
  [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] :: [Integer]
Main> let uns n | n==1 = [1] | n/=1 = 1:uns (n-1) in (sum.uns) 30
```

Funções como parâmetros

```
filtro p (x:xs)
        | p x       = x:filtro p xs
        | otherwise = filtro p xs
filtro _ [] = []                                    par x = x `mod` 2 == 0
Main> filtro par [1..21]
    [2,4,6,8,10,12,14,16,18,20] :: [Integer]
Main> filtro odd [1..20]
    [1,3,5,7,9,11,13,15,17,19] :: [Integer]
Main> map par [1..10]
    [False,True,False,True,False,True,False,True,False,True] :: [Bool]
Main> foldr (+) 0 [1..10]                           -- 55 :: Integer
```

Função como resultado

```
somat  = foldl (+) 0
produt = foldl (*) 1
append = foldr (++) []
quadlista = map (^2)
```

```
somat :: [Integer] -> Integer
produt :: [Integer] -> Integer
append :: [[a]] -> [a]
quadlista :: [Integer] -> [Integer]
```

```
Main> somat [1..10]
  55 :: Integer
Main> produt [1..10]
  3628800 :: Integer
Main> append [[1,2], [3,4],[5,6,7],[8,9,10] ]
  [1,2,3,4,5,6,7,8,9,10] :: [Integer]
Main> quadlista [1,3,5,7,9]
  [1,9,25,49,81] :: [Integer]
```

Aplicação parcial

```
Main> suc 4 where suc = (+1)
    5 :: Integer
Main> somatorio [1..10]  where somatorio = foldr (+) 0
    55 :: Integer
Main> olah "Joao"
    where olah = ("Oi " ++) . (++ " como vai voce?")
    "Oi Joao como vai voce?" :: [Char]
Main> soma 3 5 where soma = (+)
    8 :: Integer
```

Funções em Estrutura de dados

```
Main>  let operacao (f, x, y) = f x y in operacao ((+), 4, 6)
    10 :: Integer
Main> let operacao (f, x, y) = f x y
                          in map operacao [((+),4,6),((*),3,4)]
    [10,12] :: [Integer]
```

Map

```
map f xs        = [f x | x <- xs]

map2 f xs ys    = [f x y | (x,y) <- zip xs ys]
```

runhugs.exe: interpretador para scripts Hugs

runhugs *file* [*argument*...]

```
runhugs echo.hs a b c
The arguments are:
a
b
c
The program name is:
echo.hs
```

programa echo.hs

```
import System.Environment
import Data.List
main = do
   args <- getArgs
   progName <- getProgName
   putStrLn "The arguments are:"
   mapM putStrLn args
   putStrLn "The program name is:"
   putStrLn progName
```