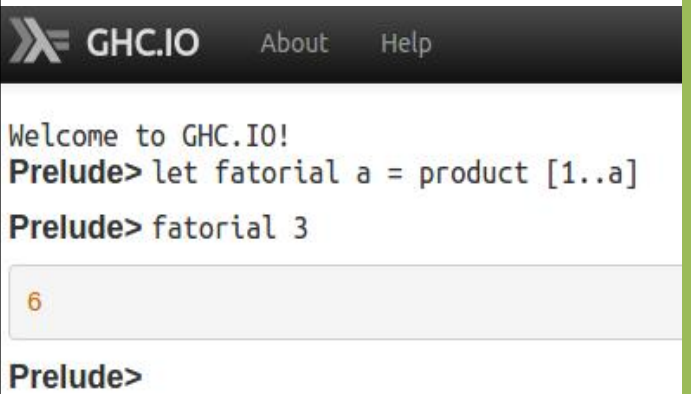

sumList [2,3,4,5]
= 2 + sumList [3,4,5]
= 2 + (3 + sumList [4,5])
= 2 + (3 + (4 + sumList [5]))
= 2 + (3 + (4 + (5 + sumList [])))
= 2 + (3 + (4 + (5 + 0)))
= 14

Programação Funcional

Funções de Alta Ordem

Matheus Carvalho Viana
matheuscviana@ufsj.edu.br

Dcomp
Departamento de Ciência da Computação

UFSJ
Universidade Federal de São João del-Rei

Introdução

As funções em Haskell podem retornar ou receber outras funções como parâmetros.

Uma função que realiza alguma dessas coisas é chamada de **função de alta ordem**.

Elas são uma forma muito poderosa de resolver problemas e pensar em programas.

O Prelude fornece diversas funções de alta ordem, principalmente, para manipulação de listas.

Funções Currying

Toda função em Haskell oficialmente recebe apenas um parâmetro.

As funções que possuem dois ou mais parâmetros de entrada são chamadas de **funções currying**:

- O primeiro parâmetro é aplicado a função e isso cria uma **função parcialmente aplicada**;
- Em seguida, essa função parcial recebe o parâmetro seguinte da função original, gerando uma nova função parcial;
- Isso se repete até todos os parâmetros serem aplicados.

Funções Currying

Considere a seguinte função:

```
multiplica :: Int -> Int -> Int -> Int
multiplica x y z = x * y * z
```

O que realmente acontece quando executamos `multiplica 2 5 9`?

1. Primeiro, 2 é aplicado à multiplica, gerando uma função “multiplica por 2” que recebe 5 e 9.
2. Então, 5 é alocado à esta, criando uma função “multiplica por 10” que recebe um parâmetro (9).
3. 9 é passado à esta última função e o resultado é 90.

Funções de Alta Ordem

Para criar uma função que recebe ou retorna uma função, basta definir como parâmetro um conjunto de parâmetros entre parênteses.

```
funAltaOrd :: (Int -> Int -> Int) -> Int -> Int -> Int
funAltaOrd f a b = f a b

filtra :: (Int -> Bool) -> [Int] -> [Int]
filtra _ [] = []
filtra f (a:b)
  | f a       = a : filtra f b
  | otherwise = filtra f b

criaSomaCom :: Int -> (Int -> Int)
criaSomaCom x = (+x)
```

Funções de Alta Ordem

Para usar uma função de alta ordem que recebe uma função, é necessário passar como entrada uma outra função que atende aos parâmetros de entrada da primeira função.

```
> funAltaOrd (+) 3 4
7
> filtra (>3) [6,1,2,3,4,8,0,5]
[6,4,8,5]
```

Para usar uma função de alta ordem que retorna uma função, é necessário usá-la em conjunto com valores de entrada válidos para a função retornada.

```
> (criaSomaCom 5) 9
14
```

Funções de Alta Ordem do Prelude

```
map :: (a -> b) -> [a] -> [b]
```

Aplica uma função a cada elemento da lista, produzindo outra lista.

```
> map (*5) [4, 9, 16]
[20,45,80]
> map (/5) [4, 9, 16]
[0.8,1.8,3.2]
> map (subtract 5) [4,9,16]
[-1,4,11]
> map length ["ab", "cde", "fghi"]
[2,3,4]
```

Funções de Alta Ordem do Prelude

```
filter :: (a -> Bool) -> [a] -> [a]
```

Aplica um teste a cada elemento da lista, produzindo outra lista somente com os elementos cujo teste resultar True.

```
> filter odd [4,5,1,0,8,7]
[5,1,7]
> filter (/=' ') "o rato roeu a roupa"
"oratoroeuaroupa"
> filter null ["", "ab", "", "cd"]
["",""]
> filter (== 0) [0,1,2,0,4,5,0]
[0,0,0]
```

Funções de Alta Ordem do Prelude

```
foldl :: (a -> a -> a) -> a -> [a] -> a
```

Aplica uma função sucessivamente, a partir dos primeiros elementos da lista (esquerda), produzindo um único resultado.

O segundo argumento é o valor inicial da operação, normalmente a identidade da mesma.

```
> foldl (+) 0 [1,2,3]
= (((0 + 1) + 2) + 3)
= ((1 + 2) + 3)
= (3 + 3)
= 6
```

Funções de Alta Ordem do Prelude

```
foldl1 :: (a -> a -> a) -> [a] -> a
```

Versão de `foldl` que não necessita de um valor inicial.

Não pode ser usado com uma lista vazia.

```
> foldl1 (+) [1,2,3]
= ((1 + 2) + 3)
= (3 + 3)
= 6
```

Funções de Alta Ordem do Prelude

`foldr` e `foldr1`

Versões de `foldl` que aplicam a operação a partir da direita.

```
> foldr (+) 0 [1,2,3]
= (1 + (2 + (3 + 0)))
= (1 + (2 + 3))
= (1 + 5)
= 6

> foldr1 (+) [1,2,3]
= (1 + (2 + 3))
= (1 + 5)
= 6
```

Composição de Funções

Na matemática, f(g(x)) significa aplicar a função g sobre x e aplicar a função f sobre o resultado da primeira.

Em Haskell, o equivalente para f(g(x)) é: **`(f . g) x`**

Funciona de forma semelhante ao pipe (|) do shell do Linux.

```
> sqrt (last [1,2,3,4])
2.0
> (sqrt . last) [1,2,3,4]
2.0
> filter (not . null) ["", "ab", "", "cd"]
["ab","cd"]
> map ((^2) . (+3)) [1,2,3,4]
[16,25,36,49]
```

Ordem dos Operandos

Operadores são infixos por padrão. Para tornar um operador prefixo, é preciso escrevê-lo entre parênteses.

Funções são prefixas por padrão. Para tornar uma função infixa, é preciso escrevê-la entre crases.

```
Prelude> (mod 2) 4
2
Prelude> (`mod` 2) 4
0
Prelude> (/2) 10
5.0
Prelude> ((/) 2) 10
0.2
```

Exercícios

1. Crie a função quicksort usando a função filter.
2. Crie uma função de alta ordem equivalente à função map.
3. Crie uma função de alta ordem equivalente à função foldl.
4. Crie uma função que recebe uma função f (`Int->Int`) e dois inteiros, x e n, e retorna uma lista [x, f x, f (f x), f (f (f x)), ...] com n elementos.
5. Crie uma função que recebe duas funções unárias f e g e uma lista e retorna uma lista em que cada elemento é f(g(x)), sendo x um elemento da lista original.
6. Crie uma função chamada zipw que recebe uma função binária f e duas listas x e y de tamanhos iguais e retorna uma lista [f x_1 y_1, f x_2 y_2, ...].
7. Na interface do GHCi, descubra como aplicar o operador == e a função mod usando composição de funções para determinar se um número é par. Faça o mesmo usando a função da questão 5.