2ª Aula Prática de PLC:
Funções de Alta Ordem em Haskell

Roteiro

- Funções de Alta Ordem
- map, filter e fold
- Lambda Cálculo
- Composição de Funções

Funções de Alta Ordem

Uma função de alta ordem é uma função que pode retornar ou receber outras funções como parâmetros.

Exemplos:

```
f :: (a -> a) -> a -> a
g :: a -> (a -> a)
h :: a -> a -> a
```

Toda função em Haskell oficialmente recebe apenas um parâmetro. As funções com mais de um parâmetro, aplicam cada um de uma vez, gerando uma nova função parcialmente aplicada.

Funções de Alta Ordem

Por exemplo:

```
soma :: Int -> Int -> Int
soma x y = x + y
```

Executar soma 4 6, por exemplo, cria primeiro uma função que recebe 4 e retorna uma função que recebe outro inteiro. Então 6 é aplicado a essa nova função e ela soma o número 6 com 4 e retorna o número 10.

```
Prelude> soma 4 6
10
Prelude> (soma 4) 6
10
```

Funções de Alta Ordem

A função criada quando aplicamos 4 é semelhante a uma nova função que recebesse apenas um parâmetro e sempre somasse o número 4 ao mesmo.

```
somaConst4 :: Int -> Int
somaConst4 y = 4 + y
```

Então, a execução de somaConst4 6 resulta no mesmo valor que a execução de soma 4 6.

```
Prelude> (soma 4) 6
10
Prelude> somaConst4 6
10
```

Funções de Alta Ordem

Observa-se então que uma função de dois ou mais argumentos pode ser aplicada parcialmente formando como resultado funções:

```
Prelude> :t soma
soma :: Int -> Int -> Int
Prelude> :t soma 4
soma 4 :: Int -> Int
Prelude> :t soma 4 6
soma 4 6 :: Int
```

Exercícios Resolvidos

1. Crie a função ordenaPeso :: (a -> Int) -> [a] -> [a], que receba uma função peso f :: (a -> Int) e ordene uma lista de elementos [a] em ordem crescente levando em consideração esse peso.

Exemplos:
Main> ordenaPeso fromEnum “ola mundo”
“ adlmnoou”
Main> ordenaPeso (*(-1)) [1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]

map, filter e fold

Principais exemplos de funções de alta ordem:

- map: transforma os elementos de uma coleção.
- filter: filtrar elementos que satisfaçam uma condição.
- fold: agrega/reduz dados dos elementos existentes.

map

A função map aplica uma função a cada elemento da lista, produzindo outra lista.

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

Exemplos:

```
Prelude> map (+5) [10, 20, 30]
[15, 25, 35]
Prelude> map fromEnum [‘a’, ‘x’, ‘y’, ‘z’]
[97, 120, 121, 122]
Prelude> map (“super” ++) [“mercado”, “amigo”, “natural”]
[“supermercado”, “superamigo”, “supernatural”]
```

filter

A função filter aplica um teste a cada elemento da lista, produzindo outra lista somente com os elementos cujo teste resultar o valor True.

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

Exemplos:

```
Prelude> filter (>5) [4, 8, 2, 16, 20]
[8, 16, 20]
Prelude> filter (<=100) (filter (>=0) [189, 12, 324, 26, -5, 6])
[12, 26, 6]
Prelude> filter even [5, 6, 7, 8]
[6, 8]
```

foldl1 e foldr1

As funções foldl1 e foldr1 aplicam uma função sucessivamente, a partir dos 2 primeiros elementos da lista (Elementos da esquerda, no caso de foldl1) ou dos 2 últimos elementos da lista (Elementos da direita, no caso de foldr1), produzindo um único resultado.

Prelude> :t foldl1 foldl1 :: (a -> a -> a) -> [a] -> a	Prelude> :t foldr1 foldr1 :: (a -> a -> a) -> [a] -> a

Exemplos:

```
Prelude> foldr1 (^) [3, 1, 2]          --   == 3^(1^(2))
3
Prelude> foldl1 (^) [3, 1, 2]          --   == ((3)^1)^2
9
```

foldl e foldr

As funções foldl e foldr são iguais as funções foldl1 e foldr1, respectivamente, entretanto, elas recebem um argumento a mais, que funciona como se fosse o valor inicial. Esse argumento será o próprio retorno caso seja passada uma lista vazia.

Prelude> :t foldl foldl :: (a -> b -> a) -> a -> [b] -> a	Prelude> :t foldr foldr :: (a -> b -> b) -> b -> [a] -> b

Exemplos:

```
Prelude> foldr (^) 4 [3, 1, 2]          --  == 3^(1^(2^(4)))
3
Prelude> foldl (^) 4 [3, 1, 2]          --  == (((4)^3)^2)^1
4096
```

Exercícios Resolvidos

2. Investigue o tipo e o funcionamento de cada uma das seguintes funções em Haskell:

a) and
b) or
c) concat
d) sum

Escreva as definições destas funções utilizando alguma função fold (foldr, foldl, foldr1 ou foldl1).

Lambda Cálculo

O λ-cálculo é uma teoria que ressurge do conceito de função como uma regra que associa um argumento a um valor calculado através de uma transformação imposta pela definição da função.

Exemplos:

```
Prelude> (\ x -> x + 5) 10
15
Prelude> (\ f x -> f (f x)) (*2) 4
16
Prelude> (\ x -> if (even x) then “Par” else “Impar”) 512
“Par”
Prelude> filter (\ x -> x>=0 && x<=100) [189, 12, 324, 26, -5, 6]
[12, 26, 6]
```

Exercícios Resolvidos

3. Utilizando obrigatoriamente as funções map, filter, o operador (++) e alguma função lambda, implemente a função classificar :: String -> String, que dada uma String contendo caracteres alfanuméricos, retorne uma nova onde os primeiros caracteres são as letras em maiúsculo da String de entrada e os últimos caracteres os números da mesma.

Exemplo:
Main> classificar “c99o9e1l32h61o6”
“COELHO999132616”

Composição de Funções

Uma forma simples de estruturar um programa é construí-lo em etapas, uma após outra, onde cada uma delas pode ser definida separadamente. Em programação funcional, isto é feito através da composição de funções, uma propriedade matemática implementada nestas linguagens que aumenta, enormemente, a expressividade do programador.

Composição de Funções

A expressão f(g(x)), normalmente, é escrita pelos matemáticos como (f.g)(x), onde o ponto (.) é o operador de composição de funções. O operador de composição é um tipo de função, cujos argumentos são duas funções e o resultado é também uma função.

```
Prelude> :t ( . )
( . ) :: (b -> c) -> (a -> b) -> a -> c
```

Composição de Funções

Exemplos:

```
Prelude> (even.(+5)) 6          --      == even((+5) 6)
False
Prelude> (head.tail) [2, 1]     --      == head(tail [2, 1])
1
Prelude> (head.tail.tail) “ski”         --      == head(tail(tail “ski”))
‘i’
```

Composição de Funções

A composição é associativa, ou seja, f . (g . h) = (f . g) . h

```
Prelude> (head.(tail.tail)) [[1, 2, 3], [3, 6, 9], [10]]
[10]
Prelude> ((head.tail).tail) [[1, 2, 3], [3, 6, 9], [10]]
[10]
```

Exercícios Resolvidos

4. Dada as composições de funções abaixo, diga, para cada uma delas, se é possível fazer a composição e o porquê (demonstrando o passo a passo).

a) tail.head
b) tail.head.head

Dados:

(.) :: (b -> c) -> (a -> b) -> a -> c

head :: [a] -> a

tail :: [a] -> [a]

Exercícios Práticos

1. A conjectura de Collatz ou conjectura do (3n + 1) pressupõe que para um número positivo inteiro qualquer n sempre pode-se chegar no número 1 seguindo dois critérios: se n for um número par, deve-se dividi-lo por 2, se n for um número ímpar, deve-se multiplicá-lo por 3 e, em seguida, somar o resultado ao número 1 para obter (3n + 1). O processo deve ser repetido por tempo indeterminado, até que, eventualmente, se chegue ao número 1.

Implemente a função: maiorSeqCollatz :: [Int] -> Int, que dada uma lista de números inteiros, retorne o tamanho da maior sequência de Collatz dentre os números positivos da mesma. Para implementar essa função, utilize obrigatoriamente as funções map, filter e alguma função fold(foldr, foldl, foldr1 ou foldl1).

Exemplos:

```
Main> maiorSeqCollatz [1, 2, 3, 4, 5]
8
Main> maiorSeqCollatz [1, -4, 5, 14, 27, 3, 0, 8, 56, -15, 9, 13]
112
Main> maiorSeqCollatz [302, 6009, -2231, 27, 553, 0, -21545]
137
Main> maiorSeqCollatz [990991, 536870906, 5555452, 98736551]
449
```

2. Determine o tipo das funções abaixo mostrando os passos até obter o resultado. Caso não seja possível determinar o tipo, explique o porquê.

a) (.) thrice map
b) swap map thrice

Dados:

```
( . ) :: (b -> c) -> (a -> b) -> a -> c

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

twice :: (a -> a) -> a -> a
twice f x = f (f x)

thrice :: (a -> a) -> a -> a
thrice f x = f (f (f x))

swap :: a -> (a -> b) -> b
swap f g = g f
```

• Os exercícios práticos devem ser realizados individualmente e enviados por e-mail com o assunto [IF686EC] EXERCÍCIOS PRÁTICOS 01 para monitoria-if686-ec-l@ cin.ufpe.br até as 23:59 de hoje (06.04.2017).

• As resoluções dos exercícios devem estar em arquivos diferentes, um arquivo por exercício, com os nomes no formato Q[número do exercício].hs.