Faculdade de Computação
Programação Funcional (BCC/BSI) - 1° Período
Aula Prática: Listas e Funções em Haskell

Seleção multi-direcional – Comando case

O comando case é utilizado quando existe a necessidade de checar múltiplos valores. Caso queira-se definir uma função que receba uma série de entradas e deva retornar um valor especial para cada entrada:

1 - “AAA”
2 - “BBB”
3 - “CCC'
outro valor - “ZZZ”

Escrever esta função utilizando **if** poderia tornar o programa de difícil leitura, sendo mais adequada a utilização da instrução **case**:

```
f x = case x of
            1 -> "AAA"
            2 -> "BBB"
            3 -> "CCC"
            _ -> "ZZZ"
```

A identação (recuo do texto do código) deve ser observada. Outra opção fornecida pela linguagem Haskell é a utilização de chaves e pontos-e-vírgulas:

```
f1 x = case x of
       { 1 -> "AAA"; 2 -> "BBB"; 3 -> "CCC"; _ -> "ZZZ" }
```

Exercícios:

1) Seja a função abaixo, definida utilizando-se o comando case. Explique seu funcionamento, faça o teste com vários tipos de entrada. Quais os tipos de entrada aceitos pela função? Mostre a passo-a-passo a execução da função para dois argumentos quaisquer.

```
import Char

strcmp s1 s2 = case (s1, s2) of
           ([], []) -> True
           (s1:ss1, s2:ss2)  | toUpper s1 == toUpper s2 -> strcmp ss1 ss2
                             | otherwise -> False
           _ -> False
```

2) Faça uma função, usando o comando case que dado um número entre 1 e 12, retorne o mês correspondente.

3) A função *tamanho* que calcula o comprimento de uma lista pode ser definida do seguinte modo:

```
tamanho xs =  tam 0 xs
     where tam n [] = n
           tam n (x:xs) = tam (n+1) xs
```

A função `tamanho` usa a função auxiliar `tam` que utiliza a variável `n` para acumular o resultado. Por sua vez, a função `tam` é definida com a técnica de recursão de cauda, uma vez que a chamada `tam (n+1) xs` não ocorre dentro de nenhuma outra função, como argumento. Para a chamada abaixo, mostre os passos da execução:

```
> tamanho "programacao funcional"
```

Em seguida, defina as funções fatorial e inverso, descritas abaixo usando recursão de cauda:

(a) `fatorialC :: Int -> Int`, para calcular o fatorial de um número natural

(b) `inversoC :: [a] -> [a]`, para retornar o inverso de uma lista

4) Avalie e teste as funções abaixo, mostrando a finalidade de cada:

```
pegar :: Int -> [a] -> [a]
pegar 0 xs = []
pegar (n+1) (x:xs) = x:pegar n xs

de :: Int -> [Int]
de n = n:de (n+1)
```

Haskell possui uma técnica de avaliação chamada *avaliação preguiçosa*, na qual não se avalia nenhuma sub-expressão até que seu valor seja reconhecido como necessário. Desta forma, mostre os passos da execução da chamada:

```
>pegar 10 (de 45)
[45,46,47,48,49,50,51,52,53,54]
```