Disciplina: Paradigmas de Programação

Alunos: Rayssa Mirelle Silva Oliveira e

Everson Eduardo da Silva Andrade

Atividade de Haskell

1. Qual é o tipo de cada uma das seguintes expressões? (Algumas delas geram erros de tipo.)
    a. `"squid" ++ "clam"`

    R: Tipo: String. A operação ++ é usada para concatenar duas strings, então o resultado será uma string, que é `"squidclam"`.

    b. `[True, False, True, True]`

    R: **Tipo: (Lista de booleanos). Esta é uma lista de valores booleanos (True e False).**

    c. `[True, False, 'a']`

    R: Tipo: Erro de tipo. As listas em linguagens fortemente tipadas como Haskell exigem que todos os elementos tenham o mesmo tipo.

    d. `(True, False, 'a')`

    R**:Tipo: (Bool, Bool, Char) (Tupla). Tuplas podem conter elementos de tipos diferentes.**

2. Escreva uma função em Haskell para encontrar o cubo de um valor do tipo `Double`. Qual é o tipo dessa função?

    R: cubo :: Double -> Double

    cubo x = x * x * x

O tipo dessa função é `Double -> Double`, o que significa que ela recebe um valor do tipo `Double` e retorna um valor também do tipo `Double`.. Seria um Float com numeros decimais maior

3. Escreva uma função em Haskell para encontrar a soma de três valores do tipo `Double`. Qual é o tipo dessa função?

   R:

   somaTres :: Double -> Double -> Double -> Double

   somaTres x y z = x + y + z

   O tipo dessa função é `Double -> Double -> Double -> Double`, o que significa que ela recebe três valores do tipo `Double` e retorna um valor do tipo `Double` como resultado da soma.

4. Escreva uma função em Haskell para encontrar o valor da expressão quadrática `ax^2 + bx + c` para os parâmetros `a`, `b`, `c` e `x`. Qual é o tipo dessa função?

   R:

   quadratica :: Num a => a -> a -> a -> a -> a

   quadratica a b c x = a * x^2 + b * x + c

   O tipo da função seria `Num a => a -> a -> a -> a -> a`, o que significa que ela funciona para qualquer tipo `a` que seja uma instância da classe `Num`. Isso inclui tipos como `Int`, `Float`, `Double`, etc.

5. Escreva uma função em Haskell para inverter uma lista. Qual é o tipo dessa função?

   R: Nesse caso podemos usar Duas formas: A primeira usando com a função `reverse` já é definida no Prelude (biblioteca padrão do Haskell) e realiza a inversão de listas de maneira eficiente. E a segunda é criando do 0 usando função recursiva:

   1-

   ```
   inverterLista :: [a] - > [a]
   inverterLista = reverse
   ```

   2-

   ```
   inverterLista :: [a] - > [a]
   inverterLista [] = []
   inverterLista (x:xs) = inverterLista xs ++ [x]
   ```

6. Faça uma função recursiva de uma lista `doubles` cujo primeiro elemento seja `10`, e cujo n-ésimo elemento seja o dobro do n-1-ésimo, ou seja, `[10, 20, 40, 80, 160, 320, ...]`.

   R:

   ```
   doubles :: [Double]
   doubles = gerarDoubles 10
     where
       gerarDoubles x = x : gerarDoubles (2 * x)
   ```

7. Defina uma lista Haskell `dollars` que seja a lista infinita dos montantes de dinheiro que você terá a cada ano, assumindo que você começa com $100 e recebe juros de 5%, compostos anualmente. (Ignore inflação, deflação, impostos, resgates, a possibilidade de colapso econômico total, e outros detalhes semelhantes.) Assim, `dollars` deve ser igual a `[100.0, 105.0, 110.25, ...]`.

   R:

   dollars :: [Double]

   dollars = iterate (\x -> x * 1.05) 100.0

8. Qual é o tipo de cada uma das seguintes expressões Haskell? (Algumas podem gerar um erro.)
   a. `my_const`:

      R: A função `const` retorna sempre o primeiro argumento, independentemente do segundo.

      ex: my_const :: a -> b -> a

      my_const x _ = x

   b. `my_const True`

      R: A função como usa a função cont, tem a mesma lógica, usado no aplicar `True` como o primeiro argumento, a função resultante sempre retornará `True` para qualquer entrada subsequente.

      Ex: -- Uso: my_const True 3

      -- Resultado: True

   c. `append`:

      R: Tipo: `[a] -> [a] -> [a]`. `append` é o operador `(++)` que concatena listas.

Ex: --

[1, 2, 3] ++ [4, 5, 6]

-- Resultado: [1, 2, 3, 4, 5, 6]

d. `append []`

R:Tipo: `[a] -> [a] -> [a]`

Ex:

`resultado6 = [] ++ [1, 2, 3]` Tipo: [Int] Resp: `[1, 2, 3]`

`resultado7 = [] ++ "Haskell"` Tipo: [Char]

e. `append [True, False]`

R:Tipo: `[Bool] -> [Bool] -> [Bool]`

Ex: `resultado8 = [True, False] ++ [True, True]` Resp:`[True, False, True, True]`

f. `append [3] ['a', 'b']`

R: **Esse exemplo causará erro de tipo, pois não é permitido concatenar listas de tipos diferentes nesse caso o (`Int` e `Char`).**

g. `append "squid" ['a', 'b']`

R: **Esse exemplo também causará erro de tipo, pois não é permitido concatenar uma string com uma lista explícita de caracteres.**

h. `my_map`

R: **Tipo:** `(a -> b) -> [a] -> [b]`

A função `my_map` é similar à função `map`, que aplica uma função a todos os elementos de uma lista:

Ex: **Uma função que multiplica cada elemento da lista por 2`(*2)`.**

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

my_map f xs = [f x | x <- xs]

Exemplo de uso:

resultado11 = my_map (*2) [1, 2, 3, 4] -- Tipo: (Int -> Int) -> [Int] -> [Int]
Resultado: `[2, 4, 6, 8]`

i. `my_map (my_const True)`

R: **Tipo: `[b] -> [Bool]`**

ex: my_map aplica a função my_const True a todos os elementos da lista, retornando uma lista de True com o mesmo comprimento da lista original

```
resultado12 = my_map (my_const True) [1, 2, 3]    -- Tipo: [Int] -> [Bool]

resultado13 = my_map (my_const True) ["apple", "banana", "cherry"]  -- Tipo: [String] -> [Bool]
```

Resp: [True, True, True] (para qualquer lista)

A função my_const True ignora o valor da lista e retorna True para cada elemento.