UDESC UNIVERSIDADE DO ESTADO DE SANTA CATARINA

Exercícios Haskell

Ciência da Computação

Estas funções já existem na biblioteca padrão do Haskell (Prelude), porém o objetivo do exercício é praticar programação funcional, implementando tais funções manualmente. Não se deve utilizar as funções padrões, porém alguns exercícios irão precisar de funções declaradas por exercícios anteriores.

1. Escreva uma função que retorne a concatenação entre duas listas.
   ```
   concatenacao :: [a] → [a] → [a]
   ```
2. Escreva uma função que retorne se um elemento pertence a uma lista.
   ```
   pertence :: Eq a => a → [a] → Bool
   ```
3. Escreva uma função que retorne a interseção entre duas listas.
   ```
   intersecao :: Eq a => [a] → [a] → [a]
   ```
4. Escreva uma função que retorne o inverso de uma lista.
   ```
   inverso :: [a] → [a]
   ```
5. Escreva uma função que retorna os *n* primeiros elementos de uma lista.
   ```
   primeiros :: Int → [a] → [a]
   ```
6. Escreva uma função que retorne os *n* últimos elementos de uma lista.
   ```
   ultimos :: Int → [a] → [a]
   ```
7. Escreva uma função que converta um número binário, representado como uma string, em um número inteiro:
   ```
   binParaInt :: String → Int
   ```
8. Escreva uma função que converta um número inteiro para um número binário, representado como uma string:
   ```
   intParaBin :: Int → String
   ```
9. Escreva uma função que retorna o menor valor de uma lista:
   ```
   menorValor :: Ord a => [a] → a
   ```
10. Escreva uma função que receba uma lista e um elemento e retorne a lista sem a primeira ocorrência desse elemento:
    ```
    removerPrimeiro :: Eq a => [a] → a → [a]
    ```
11. Escreva uma função, *utilizando os exercícios anteriores*, que ordene os elementos de uma lista:
    ```
    ordenar :: Ord a => [a] → [a]
    ```
12. Escreva uma função que dobre uma lista pela direita, lembrando que:
    dobrar_dir f x [a, b, c] = f a (f b (f c x))
    ```
    dobrar_dir :: (a → b → b) → b → [a] → b
    ```
13. Escreva uma função que dobre uma lista pela esquerda, lembrando que:
    dobrar_esq f x [a, b, c] = f (f (f x a) b) c
    ```
    dobrar_esq :: (b → a → b) → b → [a] → b
    ```
14. Escreva uma função que filtre uma lista, retornando os elementos que satisfazem um predicado:
    ```
    filtrar :: (a → Bool) → [a] → [a]
    ```
15. Usando a função anterior, escreva uma função que receba uma lista e retorne seus elementos ímpares:
    ```
    impares :: [Int] → [Int]
    ```
16. Escreva uma função que mapeie todos os elementos de uma lista de acordo com uma função:
    ```
    mapear :: (a → b) → [a] → [b]
    ```

17. Usando a função anterior, escreva uma função que receba uma lista de duplas, e retorne uma lista com os primeiros itens:
    ```
    primeiros :: [(a, b)] → [a]
    ```
18. Escreva uma função que receba uma lista de booleanos e retorne se todos são verdadeiros, utilizando uma das funções de dobra anteriores:
    ```
    todos :: [Bool] → Bool
    ```
19. Considere o tipo de dados abaixo:
    ```
    data Tree a = Leaf a
                | Branch (Tree a) (Tree a)
    ```
    Escreva uma função que encontra o maior item de uma árvore:
    ```
    maior :: Ord a => Tree a → a
    ```
20. Considere o tipo acima; escreva uma função que retorna a altura de uma árvore:
    ```
    altura :: Tree a → Int
    ```