Programação Funcional
Lista de Exercícios 02

Prof. Wladimir Araújo Tavares

1. Escreva a definição recursiva da função `concatena :: [[a]] -> [a]` que concatena uma lista de listas.

   Exemplo:

   ```
   concatena [ [1,2], [3,4] ] == [1,2,3,4]
   ```

2. Escreva a definição recursiva da função `replica :: Int -> a -> [a]` que produz uma lista com n elementos iguais.

   Exemplo:

   ```
   replica 3 1 == [1,1,1]
   ```

   ```
   replica 3 'a' == "aaa"
   ```

3. Escreva a definição recursiva da função `elemento :: Eq a => a -> [a] -> Bool` que testa se um valor ocorre numa lista

   Exemplo:

   ```
   elemento 1 [2,3,1] == True
   ```

   ```
   elemento 1 [2,3,4] == False
   ```

4. Escreva a definição recursiva da função isSorted que retorna verdadeiro se a lista está ordenada e falso, caso contrário.

   ```
   isSorted :: Ord a => [a] -> Bool
   ```

   Exemplo:

   ```
   isSorted [1,2,3,4] == True
   ```

   ```
   isSorted [2,1,3,4] == False
   ```

5. Escreva a definição recursiva da função palindromo que recebe uma string S e retorna verdadeiro se S é um palindromo e falso, caso contrário.

   `palindromo :: String -> Bool` Exemplo:

   ```
   palindromo "ana"== True
   ```

   ```
   palindromo "123a321"== True
   ```

   ```
   palindromo "cachorro"== False
   ```

   Dica: Use as funções `last:: [a] -> a` e `init :: [a] -> [a]`.

6. Escreva a definição recursiva da função rotEsq que recebe um natural n e uma lista genérica e retorna uma lista rotacionada n vezes à esquerda.

   ```
   rotEsq :: Int -> [a] -> [a]
   ```

   Exemplo:

   ```
   rotEsq 0 "asdfg"== "asdfg"
   ```

   ```
   rotEsq 1 "asdfg"== "sdfga"
   ```

   ```
   rotEsq 2 "asdfg"== "dfgas"
   ```

7. Escreva a definição recursiva da função rotDir que recebe um natural n e uma lista genérica e retorna uma lista rotacionada n vezes à direita.

   ```
   rotDir :: Int -> [a] -> [a]
   ```

   Exemplo:

   ```
   rotDir 0 "asdfg"== "asdfg"
   ```

   ```
   rotDir 1 "asdfg"== "gasdf"
   ```

   ```
   rotDir 2 "asdfg"== "fgasd"
   ```

8. Escreva a definição recursiva da função `uniao`:

   INPUT: Duas listas a e b sem repetição de chaves

   OUTPUT: Lista das chaves de a e b sem repetição

   Exemplos:

   ```
   uniao [1,2,3] [2,4,6] == [1,2,3,4,6]
   ```

   ```
   uniao [4,5] [1] == [4,5,1]
   ```

   Use a função `elemento`

9. Defina uma função recurdiva `somaDigitos :: Int -> Int` que retorna a soma dos dígitos de um número.

   - A soma dos dígitos do número zero é zero.
   - A soma dos dígitos do número n é o último dígito mais a soma dos dígitos do número formado sem o último dígito.

   Exemplo :

   ```
   somaDigitos 234 == 9
   ```

10. Escreva uma definição recursiva da função `binario :: Int -> [Int]` que calcula a representação de um inteiro positivo em algarismos binários (0 ou 1). A lista resultado deve estar ordenada do algarismo mais significativo para o menos significativo.

    Exemplo: `binario 6 = [1, 1, 0]`

11. Escreva uma função `partir :: Int -> [a] -> [[a]]` tal que partir n de decompõe uma lista em sub-listas cuja concatenação dá a lista original e tal que cada sub-lista tem comprimento n (exceto, possivelmente, a última).

    Exemplo:`partir 5 "abdefghijkl"= ["abde", "fghij", "kl"]`

    Dica: use a função `take` , `drop` e `(:)`

12. **Ordenação de listas pelo método de inserção.**

    (a) Escreva definição recursiva da função `insert :: Ord a => a -> [a] -> [a]` para inserir um elemento numa lista ordenada na posição correta de forma a manter a ordenação.

    Exemplo: `insert 2 [0, 1, 3, 5] == [0, 1, 2, 3, 5]`

    (b) Usando a função insert, escreva uma definição também recursiva da função `insertSort :: Ord a => [a] -> [a]` que implementa ordenação pelo método de inserção:

    - a lista vazia já está ordenada;
    - para ordenar uma lista não vazia, recursivamente ordenamos a cauda e inserimos a cabeça na posição correta.

13. **Ordenação de listas pelo método de seleção:**

    (a) Escreva definição recursiva da função `minimo :: Ord a => [a] -> a` que calcula o menor valor duma lista não-vazia.

    Exemplo: `minimo [5, 1, 2, 1, 3] == 1`.

    (b) Escreva uma definição recursiva da função `remove :: Eq a => a -> [a] -> [a]` que remove a primeira ocorrência dum valor numa lista.

    Exemplo: `remove 1 [5, 1, 2, 1, 3] = [5, 2, 1, 3]`.

    (c) Usando as funções anteriores, escreva uma definição recursiva da função `selectionSort :: Ord a => [a] -> [a]` que implementa ordenação pelo método de seleção:

    - a lista vazia já está ordenada;
    - para ordenar uma lista não vazia, colocamos à cabeça o menor elemento m e recursivamente ordenamos a cauda sem o elemento m.

14. Mostre como a lista em compreensão `[f x | x <-xs, p x]` se pode escrever como combinação das funções de ordem superior map e filter.
15. Defina a função sumsq que recebe um inteiro n como argumento e retorna a soma dos quadrados dos n primeiros inteiros.

    `sumsq n` $= 1^2 + 2^2 + 3^2 + \ldots + n^2$

16. Defina a função tamanho, que retorna o número de elementos em uma lista usando `foldr::(a -> b -> b) -> b -> [a] -> b` e `foldl::(b -> a -> b) -> b -> [a] -> b`.
17. Escreva a definição da função `concatena :: [[a]] -> [a]` que concatena uma lista de listas usando a função `foldr::(a -> b -> b) -> b -> [a] -> b`
18. Escreva a definição da função `inverte1 :: [a] -> [a]` que inverte uma lista usando a função `foldr:(a -> b -> b) -> b -> [a] -> b`

    Dica: Considere a seguinte definição recursiva:

    ```
    inverte :: [a] -> [a]
    inverte [] = []
    inverte xs = (last xs):inverte (init xs)
    ```

19. Escreva a definição da função `inverte2 :: [a] -> [a]` que inverte uma lista usando a função `foldl::(b -> a -> b) -> b -> [a] -> b`

    Dica: Considere a seguinte definição recursiva:

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

20. O que a função mystery faz?

    ```
    mystery xs = foldr (++) [] (map sing xs)
    sing x = [x]
    ```

21. Escreva a definição da função `elem :: Eq a => a -> [a] -> Bool` que testa se um valor ocorre em uma lista usando a função `any`.

    Dica: Crie uma lista de testes `x==y` tal que y é um elemento da lista.
22. Mostre que pode definir função `insertSort :: Ord a => [a] -> [a]` para ordenar uma lista pelo método de inserção usando foldr e insert.
23. Redefina a função `map f :: [a] -> [a]` usando a função `foldr`.
24. Redefine a função `filter p :: [a] -> [a]` usando a função `foldr`.
25. As funções `foldl1` e `foldr1` do prelúdio-padrão são variantes de foldl e foldr que só estão definidas para listas com pelo menos um elemento (i.e. não-vazias).`foldl1` e `foldr1` têm apenas dois argumentos (uma operação de agregação e uma lista) e o seu resultado é dado pelas equações seguintes.

    $foldl1(\oplus)[x_1, \ldots, x_n] = (\ldots(x_1 \oplus x_2) \oplus x_3 \ldots) \oplus x_n$

    $foldr1(\oplus)[x_1, \ldots, x_n] = x_1 \oplus (\ldots(x_{n-1} \oplus x_n)\ldots)$

    Mostre que pode definir as funções `maximo, minimo :: Ord a => [a] -> a` do prelúdio-padrão (que calculam, respectivamente, o maior e o menor elemento duma lista não-vazia) usando `foldl1` e `foldr1` .
26. Escreva a definição não-recursiva da função isSorted que retorna verdadeiro se a lista está ordenada e falso, caso contrário usando a função `all`.

    ```
    isSorted :: Ord a => [a] -> Bool
    ```

    Dica: Crie uma lista com todos os pares adjacentes (x,y) e teste se em todos os pares `x<=y` é verdadeiro.