Dúvidas e solução de exercícios

Programação Funcional

Prof. Rodrigo Ribeiro

Objetivos

- Resolução de alguns exercícios envolvendo listas.
- Dúvidas sobre o conteúdo visto até o momento.

Setup

```haskell
module Aula06 where

import Prelude hiding (take, init, zip, minimum)

import Test.Tasty
import Test.Tasty.HUnit
```

Exercício 1

- Desenvolver a função

```
take :: Int -> [a] -> [a]
take = undefined
```

que retorna um prefixo dos "n" primeiros elementos de uma lista. Sua implementação deve satisfazer os seguintes testes:

```
takeTests :: TestTree
takeTests
  = testGroup "Function take tests"
      [
        testCase "take less than list length" $
          take 3 [1..5] @?= [1,2,3]
      , testCase "take greater than list length" $
          take 5 [1,2,3] @?= [1,2,3]
      , testCase "take equal list length" $
          take 3 [1,2,3] @?= [1,2,3]
      ]
```

Exercício 2

- Desenvolver a função

```
init :: [a] -> [a]
init = undefined
```

que retorna todos os elementos da lista de entrada, exceto o último. Sua implementação deve satisfazer os seguintes testes:

```
initTests :: TestTree
initTests
  = testGroup "Function init tests"
      [
         testCase "init test" $
           init [1,2,3] @?= [1,2]
      ]
```

Exercício 3

- Desenvolver a função

```
sorted :: [Int] -> Bool
sorted = undefined
```

que determina se uma lista de inteiros fornecida como entrada está ou não ordenada. Sua implementação deve satisfazer os seguintes testes:

```
sortedTests :: TestTree
sortedTests
  = testGroup "Function sorted tests"
        [
          testCase "sorted empty" $ sorted [] @?= True
        , testCase "sorted single" $ sorted [1] @?= True
        , testCase "sorted many" $ sorted [1,2,3] @?= True
        , testCase "sorted false" $ sorted [4, 1, 5] @?= Fa
        ]
```

Exercício 4

- Desenvolver a função

```
zip :: [a] -> [b] -> [(a,b)]
zip = undefined
```

que converte duas listas em uma lista de pares.

Sua implementação deve satisfazer os seguintes testes:

```
zipTests :: TestTree
zipTests
  = testGroup "Function zip tests"
      [
        testCase "zip list same length" $
          zip [1,2] [3,4] @?= [(1,3), (2,4)]
      ,
        testCase "zip list diff length right" $
          zip [1,2] [3,4,5] @?= [(1,3), (2,4)]
      , testCase "zip list diff length right" $
          zip [1,2,3] [3,4] @?= [(1,3), (2,4)]
```

Exercício 5

- Desenvolver a função

```
minimum :: [Int] -> Int
minimum = undefined
```

que retorna o maior elemento de uma lista de números inteiros.

Sua função deve satisfazer os seguintes testes:

```
minimumTests :: TestTree
minimumTests
      = testGroup "Function mininum tests"
          [
             testCase "mininum test" $
               minimum [1, 0, 10, 5] @?= 0
          ]
```

Exercício 6

- Desenvolver a função

```
prefixes :: [a] -> [[a]]
prefixes = undefined
```

que retorna todos os prefixos de uma lista fornecida como entrada. Sua função deve satisfazer os seguintes testes:

```
prefixesTests :: TestTree
prefixesTests
   = testGroup "Function prefixes tests"
         [
           testCase "prefixes test" $
            prefixes [1,2,3] @?= [[], [1], [1,2], [1,2,3]]
         ]
```

Main

```
tests :: TestTree
tests
 = testGroup "Tests"
      [
        takeTests
      , initTests
      , sortedTests
      , zipTests
      , minimumTests
      , prefixesTests
      ]

main :: IO ()
main = defaultMain tests
```

Dúvidas?

- Espaço aberto para dúvidas sobre o conteúdo.