Avaliação 3 de Programação Funcional

ATENÇÃO

- A interpretação dos enunciados faz parte da avaliação.
- A avaliação deve ser resolvida INDIVIDUALMENTE. Não serão tolerados plágios de nenhum tipo.
- Sobre a entrega da solução:

1. Cada exercício solucionado deverá ser entregue em um arquivo jpg usando a seguinte convenção de nomes: MATRÍCULA-EXERCÍCIO.jpg. Exemplo: Se sua matrícula for 20.1.2020, o arquivo correspondente a questão 2, item b) será 2012020-2b.jpg
2. A entrega da solução da avaliação deve ser feita como um único arquivo .zip contendo imagens de suas soluções.
3. O arquivo .zip a ser entregue deve usar a seguinte convenção de nome: MATRÍCULA.zip, em que matrícula é a sua matrícula. Exemplo: Se sua matrícula for 20.1.2020 então o arquivo entregue deve ser 2012020.zip.
4. A não observância dos critério de nome e formato da solução receberá uma penalidade de 30% sobre a nota obtida na avaliação.
5. O arquivo de solução deverá ser entregue usando a atividade "Entrega da Avaliação 3" no Moodle dentro do prazo estabelecido.
6. É de responsabilidade do aluno a entrega da solução dentro deste prazo.
7. Sob NENHUMA hipótese serão aceitas soluções fora do prazo ou entregues usando outra ferramenta que não a plataforma Moodle.
8. A avaliação é formada por 4 questões, cada uma possui o valor de 2,5 pontos.

Setup inicial

```
module Main where

import Prelude hiding ((++), elem, reverse, map)
```

Provas de propriedades de programas

O objetivo dessa avaliação é provar propriedades de um programa de busca sequencial em listas.

```
elem :: Eq a => a -> [a] -> Bool
elem _ []       = False
elem x (y : ys) = x == y || elem x ys
```

As primeiras propriedades envolvem a função de concatenação de listas.

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

Nas questões a seguir, assuma que todas as listas e elementos mencionados são de um tipo `a` que possui instância da classe `Eq`, isto é, possuem implementação de funções para teste de igualdade de valores.

Questão 1. Prove o seguinte teorema: Para todas listas `xs`, `ys` e elemento `x`, se `elem x xs = True` então `elem x (xs ++ ys) = True`.

Questão 2. Prove o seguinte teorema: Para todas listas xs e ys, e elemento x, se `elem x xs = True` então `elem x (ys ++ xs) = True`.

Para a próxima questão, considere a função `reverse` que inverte os elementos de uma lista fornecida como entrada.

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

Questão 3. Prove o seguinte teorema: Para toda lista xs e elemento x, se `elem x xs = True` então `elem x (reverse xs) = True`.

Para a próxima questão, considere a função `map` que aplica uma função a cada elemento de uma lista fornecida como entrada.

```
map :: (a -> b) -> [a] -> [b]
map _ []       = []
map f (x : xs) = f x : map f xs
```

Questão 4. Prove o seguinte teorema: Para toda lista `xs :: [a]`, função `f :: a -> b` e `x :: a` temos que se `elem x xs = True` então `elem (f x) (map f xs) = True`.