Programação Funcional
Laboratório 5: Recursão e Listas

Envie seu arquivo `.hs` para bruno.welldm@gmail.com.
Assunto do email: `PF Lab 5`
Corpo do email: `nome do aluno`

1) Escreva a função recursiva `npares` que recebe uma lista de inteiros e retorna a quantidade de números pares pertencentes à lista.

2) Escreva a função recursiva `dobraPos` que recebe uma lista de inteiros `l` e retorna uma lista contendo os dobros de cada elemento de `l`.

3) Escreva a função recursiva `produtorio` que recebe uma lista de números e retorna o produto de todos os seus elementos.

4) Escreva a função recursiva `enesimo` que recebe um número inteiro n e uma lista `l` e retorna o enésimo elemento de `l`.

5) Escreva a função recursiva `fatores` a seguir que recebe um inteiro e retorna uma lista com todos os seus fatores.

```
> fatores 30
[30,15,10,6,5,3,2,1]
```

6) Escreva a função recursiva `comprime` a seguir que recebe uma lista de listas e retorna uma lista contendo todos os elementos das sublistas.

```
> comprime [[1,2],[3,4,5],[],[6]]
[1,2,3,4,5,6]
```

7) Escreva a função `tamanho` a seguir que recebe uma lista polimórfica (de qualquer tipo) e retorna a quantidade de elementos que ela possui.

```
> tamanho [1,3,5,7,9]
5

> tamanho "haskell"
7
```

8) Escreva a função `pertence` a seguir que recebe um objeto x da classe `Eq` e uma lista `l` do mesmo tipo de x, e retorna True se x pertence a `l` ou False caso contrário.

```
> pertence 8 [1,3,5,7,9]
False

> pertence 'k' "haskell"
True
```

9) Escreva a função `primeiros` a seguir que recebe um número inteiro `n` e uma lista `l` de qualquer tipo e retorna uma lista com os `n` primeiros elementos de `l`.

```
> primeiros 3 [1,3,5,7,9]
[1,3,5]

> primeiros 3 "haskell"
"has"
```

10) Escreva a função `maior` a seguir que recebe uma lista e retorna seu maior elemento.

```
> maior [4,5,2,1]
5
```

12) Escreva a função recursiva `uniaoRec` a seguir que recebe duas listas e retorna uma lista que representa a união entre as duas. Esta função elimina os elementos da 1ª lista que já estejam presentes na 2ª lista (mas mantém a ordem dos mesmos) e mantém os elementos da 2ª lista como estão.

```
> uniaoRec [1,2,3,4,5,6,7] [2,9,7,10,4]
[1,3,5,6,2,9,7,10,4]
```

13) Usando compreensão de listas, escreva a função `uniaoNRec` a seguir que faz a união de duas listas de modo que ela mantenha todos os elementos da 1ª lista na mesma ordem e no final acrescenta apenas os elementos da 2ª lista que não estejam presentes na 1ª lista.

```
> uniaoNRec [1,2,3,4,5,6,7] [2,9,7,10,4]
[1,2,3,4,5,6,7,9,10]
```

14) Escreva a função recursiva `uniaoRec2` a seguir que faz a união de duas listas de modo que ela mantenha todos os elementos da 1ª lista na mesma ordem e no final acrescenta apenas os elementos da 2ª lista que não estejam presentes na 1ª lista.

```
> uniaoRec2 [1,2,3,4,5,6,7] [2,9,7,10,4]
[1,2,3,4,5,6,7,9,10]
```