Nome: Guilherme Lucas Pereira Bernardo

LISTA DE EXERCÍCIO

1. **pertence**: recebe uma lista e um elemento qualquer e verifica se o elemento pertence à lista
   Ex.:
   pertence 1 [3,7,4,2] ==> False

```
import Text.Printf (printf)

belongs:: String -> [String] -> Bool
belongs x y = let toBool =  [z==x | z <- y] -- [True, False,
True]
                 onlyEq =  [x|x<-toBool,x] -- [True, True,
True]
                 in null onlyEq -- True or False

pertence::IO()
pertence= do arg1<-getLine::IO String -- arg 1
             arg2<-getLine::IO String -- arg 2
             let x = not $ belongs arg1 $ words arg2
             printf "%s\n" $ show x
```

2. **maior**: recebe uma lista de números e retorna o maior
   Obs.: não usar a função max.

```
maior :: [Int] -> Int
maior [x] = x
maior (x:xs)
 | maior xs > x = maior xs
 | otherwise = x
```

3. **nroOcorrencias**: recebe um elemento e uma lista qualquer, retorna o número de ocorrências do elemento na lista.
   Ex.:
   nroOcorrencias [2 1 2 4 5] ==> [(2, 2), (1, 1), (4,1), (5,1)]

```
import Text.Printf (printf)
nroOcorrencias:: Int -> [Int] -> Int
nroOcorrencias x y = let toBool =  [z==x | z <- y]
                        onlyEq =  [x|x<-toBool,x]
                        in if null onlyEq then 0 else length
onlyEq
```

4. **única_ocorrencia**: recebe um elemento e uma lista e verifica se existe uma única ocorrência do elemento na lista
   Ex.:
   unica_ocorrencia 2 [1,2,3,2] ==> False
   unica_ocorrencia 2 [3,1] ==> False
   unica_ocorrencia 2 [2] ==> True

```
import Text.Printf (printf)
unica_ocorrencia:: Int -> [Int] -> Bool
unica_ocorrencia x y = let toBool =  [z==x | z <- y]
                           onlyEq =  [x|x<-toBool,x]
                           in length onlyEq==1
```

5. **maiores_que**: recebe um número e uma lista de números, retorna uma lista com os números que são maiores que o fornecido
   Ex.:
   maiores_que 10 [4,6,30,3,15,3,10,7] ==> [30, 15].

```
maioresQue:: Int -> [Int] -> [Int]
maioresQue x y = [z | z <- y, z > x]
```

6. **remover_repetidos**: recebe uma lista e retorna outra lista sem repetição de elementos
   Ex.:
   remover_repetidos [7,4,3,5,7,4,4,6,4,1,2] ==> [7,4,3,5,6,1,2]

```
removerRepetidos :: Eq a => [a] -> [a]
removerRepetidos = rdHelper []
    where rdHelper seen [] = seen
          rdHelper seen (x:xs)
              | x `elem` seen = rdHelper seen xs
              | otherwise = rdHelper (seen ++ [x]) xs
```

7. gera_sequencia: recebe um número inteiro n positivo e retorna a lista:
   [1,-1,2,-2,3,-3, ... ,n, -n]

```
sequencia:: Int -> [Int]
sequencia n = [x | y <- [1..n], x <- [y, -y]]
```

8. **interseccao**: recebe duas listas sem elementos repetidos e retorna uma lista com os elementos que são comuns às duas
   Ex.:
   interseccao [3,6,5,7] [9,7,5,1,3] ==> [3,5,7]

```
intersect :: Eq a => [a] -> [a] -> [a]
intersect [] _ = []
intersect (x:xs) l | x `elem` l = x : intersect xs l
                   | otherwise = intersect xs l
```

9. **sequencia**: recebe dois números naturais n e m, e retorna uma lista com n elementos, onde o primeiro é m, o segundo é m+1, etc...
   Ex.:
   sequencia 0 2 ==> [] sequencia 3 4 ==> [4,5,6]

```
sequencia :: Int -> Int -> [Int]
sequencia n m = [m..(m+n)-1]
```

10. **rodar_esquerda**: recebe um número natural, uma lista e retorna uma nova lista onde a posição dos elementos mudou como se eles tivessem sido "rodados"
    Ex.:
    rodar_esquerda 0 [1,2,3,4,5] ==> [1,2,3,4,5]
    rodar_esquerda 1 [1,2,3,4,5] ==> [2,3,4,5,1]
    rodar_esquerda 3 [1,2,3,4,5] ==> [4,5,1,2,3]
    rodar_esquerda 9 [1,2,3,4,5] ==> [5,1,2,3,4]

```
rodarEsquerda :: Int -> [n] -> [n]

rodarEsquerda _ [] = []

rodarEsquerda n xs = zipWith const (drop n (cycle xs)) xs
```