Ordem superior

1. Nos exercícios a seguir, **escreva primeiramente uma função recursiva**. Em seguida, reescreva essa função utilizando função de ordem superior. **Você pode usar as funções nativas map, filter e foldr**.

   (a) Função **pares :: [Int] -> [Int]** que remove todos os elementos ímpares de uma lista

```
--recursiva
pares :: [Int] -> [Int]
pares [ ] = [ ]
pares (h:t)
        | even h = h : pares t
        | otherwise = pares t

--ordem superior sem utilizar filter
filtrar :: (a -> Bool) -> [a] -> [a]
filtrar f [] = []
filtrar f (h:t)
        | f h == True = h : filtrar f t
        | otherwise = filtrar f t

pares' :: [Int] -> [Int]
pares' l = filtrar even l

--ordem superior utilizando a função nativa
pares_2 :: [Int] -> [Int]
pares_2 = filter even l
```

   (b) Função **rm_char :: Char -> String -> String** que remove todas as ocorrências de um caractere em uma string.

```
rm_char :: Char -> String -> String
rm_char c [] = []
rm_char c (h:t)
        | c == h = rm_char c t
        | otherwise = h : rm_char c t

rm_char' :: Char -> String -> String
rm_char' c str = filtrar (/= c) str
```

   (c) Função **acima :: Int -> [Int] -> [Int]** que remove todos os números menores ou iguais a um determinado valor.

```
acima :: Int -> [Int] -> [Int]
acima x [] = []
acima x (h:t)
        | h <= x = acima x t
        | otherwise = h: acima x t

acima' :: Int -> [Int] -> [Int]
acima' x l = filtrar (> x) l
```

(d) Função **produto :: Num a => [a] -> a** que computa o produto dos números de uma lista.

```
--Recursão
produto :: Num a => [a] -> a
produto [] = 1
produto (h:t) = h * produto t

--ordem superior
reduzir :: (a -> b -> b) -> b -> [a] -> b
reduzir f z [] = z
reduzir f z (x: xs) = f x (reduzir f z xs)

produto' :: Num a => [a] -> a
produto' l = reduzir (*) 1 l  ou utilizando a função foldr  produto' l = foldr (*) 1 l
```

(e) Função **concatena :: [String] -> String** que junta uma lista de strings em uma única string.

```
concatena :: [String] -> String
concatena [] = ""
concatena (h:t) = h ++ concatena t

concatena' ::  [String] -> String
concatena' l = reduzir (++) "" l
```

2. Faça a função **ssp** que considera uma lista de inteiros e devolve a soma dos quadrados dos elementos positivos da lista.

```
ssp :: [Int] -> Int
ssp l = reduzir (+) 0 (map (^2) (filtrar (>0) l))
```

3. Defina a função **sumsq** que considera um inteiro n como argumento e devolve a soma dos quadrados dos n primeiros inteiros. Ou seja:

```
*Main > sumsq 4
30
```

pois $1^2 + 2^2 + 3^2 + 4^2 = 30$.

```
sumsq :: Int -> Int
sumsq n = reduzir (+) 0 (map (^2) [1.. n])
```

4. Uma função que devolva o valor da soma dos comprimentos de cada string (elemento) da lista. Isto é, a soma total dos comprimentos da lista de entrada.

```
somaComprimentos :: [String] -> Int
somaComprimentos l = reduzir (+) 0 (map length l)
```

5. Faça uma função que separe caracteres de números em uma string de entrada. O retorno é uma tupla, em que no primeiro argumento esteja a sequência de caracteres (string), e no segundo argumento uma sequência de inteiros. **Dica:** Utilize isAlpha e isDigit, presentes em Data.Char.
   Por exemplo:

   ```
   > separa "aA29bB71"
   ("aAbB ", "2971")
   ```

   ```
   separa :: String -> (String,String)
   separa str = (filtrar isAlpha str, filtrar isDigit str)
   ```