- A prova é individual e sem consulta.
- Qualquer tipo de cola a prova será ZERADA!
- O tempo de realização para essa prova é de 1h40min (100 minutos).
- A interpretação das questões faz parte da prova. Caso você considere que qualquer questão tenha duplo sentido, escolha uma interpretação e a apresente juntamente com sua solução.
- As questões podem ser feitas a lápis, porém, após a prova corrigida em minha sala, não é permitido mais nenhum tipo de reclamação.

Data: 10 de junho de 2019
Nome: _____ Matricula: _____

1. Defina as funções a seguir utilizando **apenas** foldr, foldl, map e filter.
   (a) `allOdd :: [Int] → [Int]`, que cria um lista contendo somente os valores impares da lista de entrada.
   `allOdd [1,2,4,6,2,1,5,6,8,9] = [1,1,5,9]`
   (b) strip :: Eq a ⇒ [a] → [a] → [a] que elimina do segundo argumento, todos os elementos do primeiro argumento.
   `strip "ask" "Haskell" = "Hell"`
   (c) concatL :: [[a]] → [a] que concatena todas as listas uma lista.
   `concatL ["um ", "dois ", "tres"] = "um dois tres"`
   (d) tamanho :: [a] → Int que retorna o tamanho da lista
2. Utilizando compreensão de lista, defina as funções:
   (a) `conta :: Eq a ⇒ a → [a] → Int` que retorna a quantidade de vezes que o primeiro argumento acontece no segundo argumento
   `conta 1 [1, 2, 3, 4, 1, 1, 2, 3, 5, 5, 1, 2, 3, 2] = 4`
   (b) `ordenado :: Ord a ⇒ [a] → Bool` que dado uma lista, retorna verdadeiro se a lista esta ordenada.
   `ordenado [1, 2, 5, 7] = True`
   `ordenado [3, 3, 55, 7] = False`
   (c) tamanho :: [a] → Int que retorna o tamanho da lista
   `tamanho [1, 2, 3] = 3`
3. Seja uma árvore binária de busca representada da seguinte maneira:

```
data Tree a = Leaf
            | Node (Tree a) a (Tree a)
      deriving Show
```

   Crie as funções que façam:
   (a) Adicione um elemento a árvore
   (b) Compute o tamanho da árvore
   (c) Aplica uma função em todos os elementos de uma árvore
   `mapT :: (a → b) → Tree a → Tree b`

```
div, mod ::  Integral a => a -> a -> a
even, odd ::  Integral a => a -> Bool
(+), (*), (-), (/) ::  Num a => a -> a -> a
(<), (<=), (>), (>=) ::  Ord => a -> a -> Bool
(==), (/=) ::  Eq a => a -> a -> Bool
(&&), (||) ::  Bool -> Bool -> Bool
not ::  Bool -> Bool
max, min ::  Ord a => a -> a -> a
isAlpha, isLower, isUpper, isDigit ::  Char -> Bool
toLower, toUpper ::  Char -> Char
ord ::  Char -> Int
chr ::  Int -> Char
```

(a) Funções básicas

```
sum, product ::  (Num a) => [a] -> a
sum [1.0,2.0,3.0] = 6.0
product [1,2,3,4] = 24

maximum, minimum ::  (Ord a) => [a] -> a
maximum [3,1,4,2] = 4
minimum [3,1,4,2] = 1
concat ::  [[a]] -> [a]
concat ["go","od","bye"] = "goodbye"

(!!)  ::  [a] -> Int -> a
[9,7,5] !!  1 = 7

head ::  [a] -> a
head "goodbye"= 'g'

init ::  [a] -> [a]
init "goodbye"= "goodby"

takeWhile ::  (a->Bool) -> [a] -> [a]
takeWhile isLower "goodBye"= "good"

dropWhile ::  (a->Bool) -> [a] -> [a]
dropWhile isLower "goodBye"= "Bye"

elem ::  (Eq a) => a -> [a] -> Bool
elem 'd' "goodbye"= True

repeat ::  a -> [a]
repeat 2 = [2,2..]
repeat 'x' ['x',..]

zip ::  [a] -> [b]-> [(a,b)]
zip [1,2,3] [1,4] = [(1,1),(2,4)]

map ::  (a -> b) -> [a] -> [b]
map (^2) [1,4,9] = [1,16,81]

foldr ::  (a -> b -> b) -> b -> [a] -> b
foldr (^) 1 [2,3,2] = 512
```

```
and, or ::  [Bool] -> Bool
and [True,False,True] = False
or [True,False,True] = True

reverse ::  [a] -> [a]
reverse "goodbye"= "eybdoog"

(++) ::  [a] -> [a] -> [a]
"good"++ "bye"= "goodbye"

length ::  [a] -> Int
length [9,7,5] = 3

tail ::  [a] -> [a]
tail "goodbye"= "oodbye"

last ::  [a] -> a
last "goodbye"= 'e'

take ::  Int -> [a] -> [a]
take 4 "goodbye"= "good"

drop ::  Int -> [a] -> [a]
drop 4 "goodbye"= "bye"

replicate ::  Int -> a -> [a]
replicate 5 '$' = "$$$$$"

group ::  Eq a => [a] -> [[a]]
group [1,1,2,4,4] =[[1,1],[2],[4,4]]

zipwith ::  (a -> b -> c) -> [a] -> [b] -> [c]
zipwith (+) [1,3] [5,6] = [6,9]
```

(b) Outras funções de bibliotecas Haskell