

1. Utilizando *list comprehension*, gere uma expressão que calcule $1^2 + 2^2 + \dots 100^2$.
2. De maneira similar à função *length*, mostre como a função *replicate* :: Int -> a -> [a], que retorna uma lista de elementos idênticos, pode ser definida utilizando *list comprehension*.

```
{-exemplo}
Main> replicate 3 True = [True, True, True]
```

3. Uma tupla (x, y, z) de número inteiros é pitagórica se $x^2 + y^2 = z^2$. Utilizando *list comprehension*, defina a função *pyths* :: Int -> [(Int, Int, Int)] que, dado um limite, retorne todas as tuplas de (x, y, z) que são pitagóricas até o limite fornecido.

```
{-exemplo}
Main> pyths 10 = [(3,4,5), (4,3,5), (6,8,10), (8,6,10)]
```

4. Um inteiro positivo é perfeito se é igual à soma dos seus fatores, excluindo ele próprio. Utilizando *list comprehension*, defina a função *perfects* :: Int -> [Int] que retorna a lista de todos os números perfeitos até o limite fornecido.

```
{-exemplo}
Main> perfects 500 = [6, 28, 496]
```

5. Mostre como a seguinte *list comprehension* [(x,y) | x <- [1,2,3], y <- [4,5,6]], com dois geradores, pode ser reescrita utilizando duas *list comprehensions* contendo um único gerador. Dica: utilize a função de concatenação e concatene uma à outra.

```
Main> [(x,y) | y <- [4,5,6] | x <- [1,2,3]] =
[(1,4), (1,5), (1,6)], [(2,4), (2,5), (2,6)], [(3,4), (3,5), (3,6)]
```

6. Defina a função *find* utilizada na função *positions*.

```
positions :: Eq a => a -> [a] -> [Int]
positions x xs = find x (zip xs [0..n])
  where n = (length xs) - 1
```

7. O produto escalar de duas listas de inteiros `xs` e `ys`, de tamanho `n`, é dado pela soma do produto dos inteiros correspondentes.

```
n=1
-
\ (xsi * ysi)
/
-
i=0
```

Mostre como a função `scalarproduct :: [Int] -> [Int] -> Int`, que retorna o produto escalar de duas listas, pode ser definida utilizando `list comprehension`.

```
{-exemplo-}:
Main> scalarproduct [1,2,3] [4,5,6] = 32
```

8. Defina o operador de exponenciação `&!` para inteiros não negativos, utilizando o mesmo padrão de recursividade do operador de multiplicação. Mostre como `2 &! 3` é calculado utilizando a função que você definiu.
9. Mostre como a seguinte *list comprehension* `[f x | x <- xs, p x]` pode ser reescrita utilizando funções de alta-ordem como *map* e *filter*. Tente entender e aplicar o seguinte exemplo:

```
Main> [(+7) x | x <- [1..10], odd x].
```

10. Defina a função `dec2int :: [Int] -> Int` que converta uma lista de inteiros para um inteiro.

```
{-exemplo}
Main> dec2int :: [2,3,4,5] = 2345
```

11. A função de alta-ordem `unfold` que retorna uma lista pode ser definida como:

```
unfold p h t x
  | p x = []
  | otherwise = h x : unfold p h t (t x)
```

Com a chamada da função `unfold`, crie uma lista das potências de 2 com limite = 2^{10} .

12. Defina a função `evenCubes :: Int -> [Int]` que, dado um limite, retorne a lista do cubo dos números pares até o limite fornecido.

```
{-exemplo}
Main> evenCubes 10 = [8, 64, 216, 512]
```

13. Utilizando *list comprehension*, defina a função `insertOrd :: Int -> [Int] -> [Int]` que, dada uma lista de inteiros ordenada, insere na lista o parâmetro passado mantendo a lista ordenada.

```
{-exemplo}
Main> insertOrd 4 [0,1,2,5,6] = [0,1,2,4,5,6]
```

14. Questão 9 - Lista 1 (Nova proposta): Escreva, em *Haskell*, uma função que retorna quantos múltiplos de um determinado inteiro tem em um intervalo fornecido utilizando *list comprehension* e funções de alta ordem. Por exemplo, o número 4 tem 2 múltiplos no intervalo de 1 a 10.

```
howManyMultiples 4 1 10 = 2
```

15. Questão 19 - Lista 1 (Nova proposta): Implemente a função `duplicate::String ->Int->String` que recebe uma string *s* e um número inteiro *n*, utilizando *list comprehension* e funções de alta ordem. A função deve retornar a concatenação de *n* cópias de *s*. Se *n* for zero, retorna `.` Como dica, usar o operador de concatenação pré-definido `(++)::String->String->String`.
16. Questão 20 - Lista 1 (Nova proposta): Implemente a função `pushRight::String->Int->String`, que recebe uma string *s* e um número inteiro *n* e retorna uma nova string *t* com *k* caracteres `'>'` inseridos no início de *s*, utilizando *list comprehension* e funções de alta ordem. O valor de *k* deve ser tal que o comprimento de *t* seja igual a *n*. Obs: se *n* é menor que o comprimento de *s*, a função retorna a própria string *s*.

```
{-exemplo-}
Main> pushRight "abc" 5 = ">>abc"
```

17. Questão 22 - Lista 1 (Nova proposta): Faça em *Haskell* uma solução para inverter os elementos de uma lista de Inteiros utilizando *list comprehension* e funções de alta ordem.

```
{-exemplo-}
Main> inverte [1,2,3,4,5,6,150] = [150,6,5,4,3,2,1]
```

18. Questão 23 - Lista 1 (Nova proposta): Faça em *Haskell* uma solução para, dada uma lista de inteiros, retornar uma dupla de listas de inteiros onde a primeira conterá os elementos ímpares e a segunda os elementos pares passados como parâmetro. Utilize obrigatoriamente *list comprehension*.

```
{-exemplo-}  
Main> separa [1,4,3,4,6,7,9,10] = ([1,3,7,9],[4,4,6,10])
```

19. Questão 24 - Lista 1 (Nova proposta): Faça em *Haskell* uma solução para, dada uma lista de inteiros, retornar a string contendo as letras do alfabeto cuja posição é dada pelos elementos da lista. Utilize *list comprehension* e, caso necessário, funções de alta ordem.

```
{-exemplos-}  
Main> converte [1,2,6,1,9] = "ABFAI"  
Main> converte [ ] = "".
```

20. Questão 26 - Lista 1 (Nova proposta): Faça em *Haskell* uma solução para o seguinte problema utilizando *list comprehension* e/ou funções de alta ordem: Dada uma lista de caracteres `[Char]`, e um caractere *a*, retornar quantos caracteres da lista são iguais a *a*.

```
{-exemplo-}  
Main> conta "ABCAABCDDA" "B" = 2
```

- (a) Questão 28 - Lista 1 (Nova proposta): Faça uma solução em *Haskell* utilizando *list comprehension* que, dada uma lista de inteiros, ela retorne uma lista com uma repetição de cada elemento de acordo com seu valor.

```
{-exemplo-}  
Main> proliferaInt [3,0,2,4,0,1] = [3,3,3,2,2,4,4,4,4,1]
```

21. Questão 29 - Lista 1 (Nova proposta): Faça uma solução em *Haskell* que, dada uma lista de caracteres maiúsculos, ela retorne uma lista com uma repetição de cada elemento de acordo com o valor de sua ordem no alfabeto. Faça a solução utilizando *list comprehension*.

```
{-exemplo-}  
Main> proliferaChar [C,B,D] = "CCCBDDDDD"
```

22. Compare as seguintes implementações de uma função que verifica a existência de um elemento na lista e responda:

```
procuraElemento :: Int -> [Int] -> Bool
procuraElemento n (x:xs) = n == x || procuraElemento n xs

procuraElemento2 :: Int -> [Int] -> Bool
procuraElemento2 n (x:xs) = procuraElemento n xs || n == x
```

- (a) Ambas as implementações estão corretas para o problema em questão? Se sim, qual a diferença existente na computação das duas funções?
- (b) Qual implementação é mais eficiente?
23. Dada as seguintes funções *checkEqual* e *allEqual*, que servem para verificar se todos os elementos de uma lista são iguais, responda ao que se pede:

```
checkEqual :: Eq a => a -> [a] -> Bool
checkEqual _ [] = True
checkEqual y (z:zs) = (y == z) && checkEqual y zs

allEqual :: Eq a => [a] -> Bool
allEqual [] = True
allEqual (x:xs) = checkEqual x xs
```

- (a) Explique o porquê de ambas as funções precisarem da classe *Eq*
- (b) O que aconteceria se a restrição *Eq* fosse removida das assinaturas das funções?
24. Em *Haskell*, além da classe *Eq*, também existe a classe *Ord*. Explique a sua importância na implementação de funções e como ela pode ser utilizada.
25. Dada a seguinte implementação de árvore binária:

```
data Arvore a = Nulo | Folha a | No a (Arvore a) (Arvore a)
```

e a seguinte imagem, responda ao que se pede:

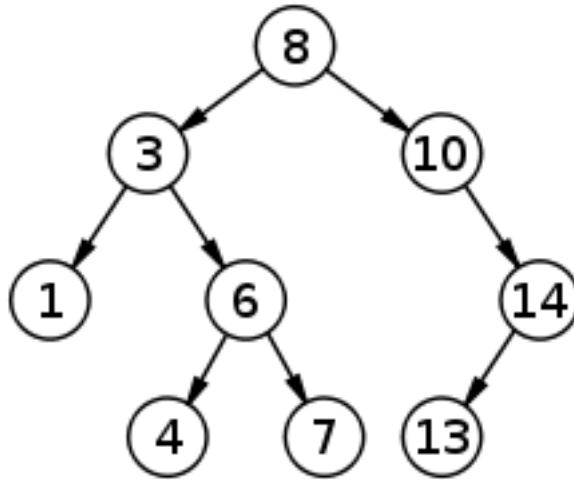


Figura 1: árvore binária

- (a) Implemente a função `emOrdem::Arvore a->[a]` que, dada uma árvore a , retorne uma lista com seus elementos em ordem crescente.

`{-exemplo-}`

```
Main> emOrdem (No 12 (No 4 (Folha 2) (No 8 (Folha 6) Nulo)) (Folha 16))
retorna: [2,4,6,8,12,16]
```

- (b) Faça a chamada da função `emOrdem::Arvore a->[a]` para a árvore da Figura 1.
- (c) Implemente a função `insere::(Ord a)=>Arvore a->n->Arvore a` que, dado um número inteiro n e uma árvore a , insere esse inteiro n corretamente na árvore a
- (d) Implemente a função `posOrdem::Arvore a->[a]` que, dada uma árvore a , retorne uma lista com seus elementos em pós ordem.

`{-exemplo-}`

```
Main> posOrdem (No 12 (No 10 (Folha 8) Nulo) (No 15 (Folha 14) (Folha 16)))
retorna: [8, 10, 14, 16, 15, 12]
```

Referências

- Exercícios 1 ao 11: Hutton, Graham. Programming in Haskell. Seção de Exercícios.

Bom Trabalho!

eliseu César miguel

Texto elaborado em L^AT_EX. Seja Livre! Seja Legal!