Exercícios Finais

1. Use **foldr** e notação **lambda** para definir uma função inverte que percorre uma lista qualquer e retorna essa lista com seus elementos invertidos.

Main> inverte ”madrom em socacam”
”macacos me mordam”

2. Defina uma função chamada **palavra** que recebe um número inteiro não negativo como entrada e imprime seus algarismos por extenso separados por hífen. Utilize a definição de tipos.

Main>palavra 175
"um-sete-cinco"

3. Defina uma função ordenada para verificar se uma lista de inteiros se encontra ordenada.

Main> ordenada [1,2,3]
True

4. Crie uma função que determine o maior divisor comum entre dois números.

5. Defina um novo tipo de dados **Point** com um construtor **Pt** que aceita dois Floats como argumento, correspondendo às coordenadas x e y de um ponto. Escreva uma função inc que aceita um Point e retorna um novo Point cujas coordenadas são incrementadas em 1.

6. Implemente função **toInts** que pega um número na forma de uma string e retorna uma lista de seus dígitos como valores inteiros.

7. Reutilize a função **toInts** para criar a função **sumDig** que pega um número na forma de uma string e retorna a soma de seus dígitos.

8. Defina uma função chamada **largestN :: Int → [Int] → [Int]**, que recebe um número **n** e uma lista de inteiros como entrada e retorna uma lista do dobro dos números que são maiores que n. Utilize **map** e **filter.**

9. Escreva uma função **quadSomaNPrimos :: [String]** → **Int**, que recebe uma lista de strings como entrada. Cada caractere é convertido para seu valor da tabela ASCII e seus valores somados, resultando em uma lista de inteiros, onde são verificados se são números primos. Então retorne a soma do quadrado dos números que não são primos. Utilize map e filter e não utilize length ou sum.

10. Usando uma expressão lambda, a função booleana **not** e a função pré-definida **elem**, defina uma função do tipo **Char -> Bool** que retorna **True** apenas para caracteres que não são espaço em branco, ou seja, aqueles que não são elementos da lista " \t\n".

11. Definir a função **oneLookupFirst :: Eq a => [(a,b)] -> a -> b**, que recebe uma lista de pares e um item e retorna a segunda parte do primeiro par, cuja primeira parte é igual ao item. Você deve explicar o que sua função faz se não houver tal par.

```
Main> oneLookupFirst [(1, 2), (23, 3), (1, 13)] 23
3
```

12. Considerando a classe a seguir, como você faria os tipos Bool, par (a, b) e tripla (a, b, c) serem da classe Visible (defina as instâncias)?

```
class Visible a where
    tostring : : a -> String
    size : : a -> Int
```

13. Definir uma função **composeList :: [a -> a] -> a -> a** que compõe uma lista de funções em uma única função.

```
Main> composeList [\x -> x+1, \x -> 2 * x, id] 3
7
```