(1) Implementar a função

`multLista :: Int -> [Int] -> [Int]`

que recebe um inteiro e uma lista e multiplica todos os elementos dessa lista pelo inteiro

(2) Implemente a função

`elemento :: Int -> [Int] -> Bool`

que recebe um inteiro e uma lista, e devolve um booleano dizendo se o inteiro se encontra na lista

(3) Implemente a função

`conta :: Int -> [Int] -> Int`

que recebe um inteiro e uma lista, e diz quantas vezes o inteiro ocorre dentro da lista

(4) Implemente a função:

`contaMaiores :: Int -> [Int]-> Int`

que recebe um inteiro e uma lista e conta quantos elementos da lista são maiores que o inteiro passado como argumento

(5) Implemente a função

`maiores :: Int -> [Int] -> [Int]`

que recebe um inteiro e uma lista e devolve uma lista contendo somente os valores que estavam na lista inicial e que são maiores do que o inteiro passado como argumento

(6) Implementar a função

`geraLista :: Int -> Int -> [Int]`

que recebe um inteiro `m` e um inteiro `n` e devolve uma lista contendo `m` vezes `n`

```
> geraLista 3 7
[7, 7, 7]
```

(7) Implementar a função

`addFim :: Int -> [Int] -> [Int]`

que recebe um inteiro, uma lista e adiciona o elemento no fim da lista (sem usar o ++):

```
> addFim 10 [1,2]
[1,2,10]
```

(8) Implementar a função

`join :: [Int] -> [Int] -> [Int]`

que recebe duas listas e concatena as mesmas gerando uma nova lista:

```
> join [1,2,3] [4,5,6]
[1,2,3,4,5,6]
```

(9) Implementar a função inverte

`inverte :: [Int] -> [Int]`

que recebe uma lista e devolve a mesma invertida