Introduction to Functional Programming with Haskell
Exercise Sheet 2

1 Exercises

1. Complete the function

```
takeWhile' ::  (A -> Bool) -> [a] -> [a]
takeWhile' f [] = ...
takeWhile' f (x:xs) = ...
```

   where `takeWhile' f xs` returns elements of `xs` as a list until it reaches an element of `xs` for which `f` is false. For example, `takeWhile' (\x -> x < 3) [1, 2, 3, 4]` returns `[1, 2]`. Look at the implementation of `filter` in the course notes for hints.

2. Recalling the `BinTree` datatype:

```
data BinTree a = Leaf a | Node a (BinTree a) (BinTree a)
```

   write the functions `size :: BinTree a -> Int`, which returns the number of labels in a tree, and `flatten :: BinTree a -> [a]`, which returns the labels of a tree as a list. You will need to provide a case for each constructor, `Leaf` and `Node`, so your functions will have this structure:

```
f (Leaf x) = ...
f (Node x t1 t2) = ...
```