In class this week we introduced a number of functions dealing with lists and iterating over and computing values with lists. For instance, length.

\[
\text{length} :: [a] \rightarrow \text{Int}
\]

\[
\text{length} \ [\ ] = 0
\]

\[
\text{length} \ (x:xs) = 1 + (\text{length} \ xs)
\]

And of course map, which takes a function and a list. It iterates over the list applying the function to each element of the list and creating a new list by “cons-ing” that element onto a new list.

\[
\text{map} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]
\]

\[
\text{map} \ f \ [\ ] = [\ ]
\]

\[
\text{map} \ f \ (x:xs) = (f \ x) : (\text{map} \ f \ xs)
\]

In this homework you are to write some functions using the defined functions from class (or if available use the prelude definitions) that meet some example specifications.

1. Write a function called firstMap, using the map function, that will take a list of pairs and return a list containing only the first element of the pairs. Write a second function called secondMap that similarly extracts a list of the second elements in the pairs.

   ex: firstMap [(1,'a'), (2,'b')] = [1,2]

2. Write a function called removeElem, which uses the member function defined in class or the elem function from the prelude.

   removeElem :: a -> [a] -> [a]

   removeElem will test each item in the list to see if it is equal to an argument. If it is remove it from the list. All other elements should remain in the list and in order.

   ex: removeElem 'a' ['v','w','a','t'] = ['v','w','t']

3. Write a function called pairwiseOp. pairwiseOp should take two lists of integers (of the same length, otherwise return the empty list) and an operator over integers. The operator should combine an element from the first list and from the second list into one element stored in a new list.

   ex pairwiseOp (+) [1,2] [3,4] = [4,6]

4. Write a function that combines firstMap and secondMap with pairwiseOp to generate a list of integers. In this case you will be given a list of pairs (both elements being integers) and the operator.

5. Define reverse in terms of foldr. You can use the following operator:

   \[
   \text{snoc} \ x \ xs = xs ++ [x]
   \]

   (basically cons backwards)
6. Write a function called `find`::\( a \to [(a,b)] \to b \) which takes an element, locates the element in a list of pairs and returns the second element if it matches. If the element is not found return an exception (error…)

7. Write `partition`::\((a \to \text{Bool}) \to [a] \to ([a],[a])\). The functionality of partition is to take a predicate and a list of elements and return a pair of lists. The first list of the pair should be where the predicate returns True and the second where it is false.

   ex: `partition` (isEven) [0..9] = ([0,2,4,6,8], [1,3,5,7,9])
   `partition` (\(x \to x > 5\)) [0..9] = ([6,7,8,9],[0,1,2,3,4,5])