Homework 4

CS3490: Programming Languages

Due: Monday, 2021.09.19–23:00

Read Chapter 6 of the Haskell tutorial at

`http://learnyouahaskell.com/higher-order-functions`

Implement the following functions using higher-order constructs and combinators introduced in that chapter or in class — composition, application, flip, etc.

1 Higher-order: first functions

Give a one-line definitions of each function in this section.
Do not use recursion. Do not use pattern-matching.

1.1 Currying and uncurrying

1. Write a higher-order function `mapPair` that takes a binary function of type `a -> b -> c` and *a list of pairs* `[(a,b)]`, and outputs a list of type `[c]`, obtained by applying the function to the two components of each pair.

```
mapPair :: (a -> b -> c) -> [(a,b)] -> [c]
mapPair (-) [(1,2),(10,20),(0,1),(-2,4)] = [-1,-10,-1,-6]
mapPair take (zip [1..] ["hi","there","string","list"]) = ["h","th","str","list"]
```

2. Write a variant of the above function which can handle the list having opposite types to the binary function

```
mapPair' :: (a -> b -> c) -> [(b,a)] -> [c]
mapPair' (-) [(1,2),(10,20),(0,1),(-2,4)] = [1,10,1,6]
mapPair' take (zip ["hi","there","string","list"] [1,3,2,2]) = ["h","the","st","li"]
```

1.2 zipWith

1. Use `zipWith` to define a function `diff` which takes two lists of integers, and subtracts each element of the first list from the corresponding element of the second.

2. Use `zipWith` invoked with the right lambda expression to define a function `splice` which takes two lists of strings, and inserts every element of the first list between two copies of the corresponding element of the second list.

```
diff [10,20,30] [5,6,7] = [5,14,23]
diff [5,6,7] [1,11,111] = [4,-5,-104]
splice ["blue","green","high"] [" sky is "," grass ","-school-"]
  = ["blue sky is blue","green grass green","high-school-high"]
```

1.3 map

1. Use `map` to define a function `sqLens :: [String] -> [Integer]` which takes a list of strings, and produces a list giving their lengths, squared.
   If necessary, you may use `fromIntegral` to convert between `Int` and `Integer`.

2. Use `map` to define a function `bang` which takes a list of strings, and attaches an exclamation point `!` to the end of each.

```
sqLens ["This", "is","a","list","of","strings"] = [16,4,1,16,4,49]
bang ["Hello","World"] = ["Hello!","World!"]
```

1.4 filter

1. Use `filter` to define a function `digitsOnly` which takes a list of *integers*, and removes all the numbers which are negative, or greater than 9.

2. Use `filter` to define a function `removeXs` which takes a list of *strings* and removes all strings which begin with the character 'X'.

```
digitsOnly [3,5,-1,12,4,21] = [3,5,4]
removeXs ["ABC","XYZ","32","","X32"] = ["ABC","32",""]
```

2 Higher-order: using folds

For each function `fun` in the following list, provide *two* implementations.
The first, called `fun`, should use recursion and patern-matching.
The other, called `fun'`, should use a single fold and neither recursion nor pattern-matching.

1. `findNum :: Integer -> [Integer] -> Bool` takes a number `n`, a list of integers `xs`, and returns `True` if `n` occurs somewhere in `xs`, and returns `False` otherwise.
2. `exists :: (a -> Bool) -> [a] -> Bool` takes a predicate on a type `a` and a list of elements of type `a`. It returns `True` if there exists an element in the list satisfying the predicate, `False` otherwise.
3. `noDups :: Eq a => [a] -> [a]` takes a list and removes all duplicate elements, leaving a list in which the same elements occur at most once.
4. `countOverflow :: Integer -> [String] -> Integer` takes an integer `x`, a list of strings, and returns the number of strings in the list whose lengths are greater than `x`.
5. `concatList :: [[a]] -> [a]` takes a list of lists and produces a single list resulting from appending all the sublists together.
6. `bindList :: (a -> [b]) -> [a] -> [b]` combines the behavior of `concatList` and `map`: it takes a function that maps *elements* of type `a` to *lists* of a potentially different type `[b]`, and then returns a list in which the function is applied to every input element, and the resulting lists are concatenated together.

```
findNum 0 [1,2,3,4,5] = False
findNum 4 [1,2,3,4,5] = True
exists even [1,3,5,7,9] = False
exists (\x -> x == 'e') "Hello" = True
noDups [1,2,2,3,3,3] = [1,2,3]
noDups "Hello world" = "Helo wrd"
countOverflow 4 ["Hello","again","this","and","that"] = 2
countOverflow 2 ["Hello","again","this","and","that"] = 5
concatList [[1,2,3],[4],[5,6]] = [1,2,3,4,5,6]
concatList ["List","Of","Strings"] = "ListOfStrings"
bindList show [123,456,789] = "123456789"
bindList (\x -> [x,x]) [1,2,3,4,5] = [1,1,2,2,3,3,4,4,5,5]
```