19CSE313

Principles of Programming Languages

Lab 8

S Abhishek

AM.EN.U4CSE19147

1 - Use foldr and foldl to define functions lengthr and lengthl respectively to find the number of elements in a list.

```
lengthr :: [Int] -> Int

lengthl :: [Int] -> Int

lengthr = foldr (\x y -> 1 + y) 0

lengthl = foldl (\x y -> x + 1) 0
```

```
*Main> lengthr [1,2,3,4,5]
5
*Main> lengthl [1,2,3,4,5]
5
*Main> lengthl [1,2,3,4]
4
*Main> lengthr [1,2,3,4]
4
```

2 - Find the smallest element of a list by defining functions minr and minl with the implementation of foldr and foldl respectively.

minr :: [Int] -> Int

minl :: [Int] -> Int

minr = foldr (\x y -> if x < y then x else y) 1000000

minl = foldl (\x y -> if x < y then x else y) 1000000

```
*Main> minr [4,7,8,9,10]
4
*Main> minl [0,-1,6,-10,3,-2,-11]
-11
*Main> minr [0,-1,6,-10,3,-2,-11]
-11
*Main> minl [4,7,8,9,10]
4
```

3 - Using foldr, define a function to reverse the current list

rev :: [Int] -> [Int]

rev = foldr (\x y -> y ++ [x]) []

```
*Main> rev [1,2,3,4,5]
[5,4,3,2,1]
*Main> rev [1,2,3]
[3,2,1]
*Main> rev [1,0,1,1]
[1,1,0,1]
*Main> rev [1]
[1]
```

4 - Define a function remover using foldr which takes two strings as its arguments and removes every letter from the second list that occurs in the first list.

```
remove :: String -> String -> String

remove str1 str2 = foldr(\x y -> if elem x str1 then y else x : y) "" str2
```

```
*Main> remove "ece" "cse"
"s"
*Main> remove "cse" "ece"
""
*Main> remove "abhi" "abhishek"
"sek"
```

5 - Remove adjacent duplicates from a list.

```
rmDup :: Eq a => [a] -> [a]

rmDup [] = []

rmDup [x] = [x]

rmDup (x:y:xs) | x == y = rmDup(y:xs)

 | otherwise = x : rmDup(y : xs)



joinr :: Eq a => a -> [a] -> [a]

joinr x [] = [x]

joinr x xs | x == head xs = xs

 | otherwise = [x] ++ xs
```

```
rmdFoldr :: Eq a => [a] -> [a]

rmdFoldr [] = []

rmdFoldr ys = foldr joinr [] ys



joinl :: Eq a => [a] -> a -> [a]

joinl [] x = [x]

joinl xs x | last xs == x = xs

 | otherwise = xs ++ [x]



rmdFoldl :: Eq a => [a] -> [a]

rmdFoldl ys = foldl joinl [] ys
```

```
*Main> rmDup [1,1,1,2,2,3,4,5,6,7,7,8]
[1,2,3,4,5,6,7,8]
*Main> rmdFoldr [1,1,1,2,2,3,4,5,6,7,7,8]
[1,2,3,4,5,6,7,8]
*Main> rmdFoldl [1,1,1,2,2,3,4,5,6,7,7,8]
[1,2,3,4,5,6,7,8]
*Main> rmdFoldl [1,1,1]
[1]
*Main> rmdFoldl [1,2,3,1,1]
[1,2,3,1]
```

6 - Define a function approxe n using foldl

$$approxe\ n = \sum_{i=0}^{i=n} \frac{1}{i!}.$$

For example,

$$\begin{aligned} approxe\ 4 &= \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!}, \\ &= 1 + 1 + 0.5 + 0.\dot{3} + 0.25, \\ &= 3.08\dot{3}, \end{aligned}$$

where $0.\dot{3}$ means 'point 3 recurring'.

```
approx :: Float -> Float

approx n = foldl findApprx 1 [1 .. n] where findApprx x y = x + (1/product[1 .. y])
```

```
*Main> approx 4
2.7083335
*Main> approx 3
2.6666667
*Main> approx 1
2.0
*Main> approx 7
2.718254
```

7 - Define the function mult using lambda expressions

```
mult :: Num a => a -> a -> a -> a

mult a b c = (\x y z -> x * y * z) a b c
```

```
*Main> mult 2 3 4
24
*Main> mult 10 1 10
100
*Main> mult 10 1 5
50
*Main> mult 5 5 5
125
*Main> mult 10 10 10
1000
```

8 - Define the function add using lambda expressions.

```
add :: Num a => (a,a) -> (a,a) -> (a,a)

add (a,b) (c,d) = (\x0 y0 x1 y1 -> (x0+x1, y0+y1)) a b c d
```

```
*Main> add (1,2) (3,4)
(4,6)
*Main> add (1,1) (0,0)
(1,1)
*Main> add (1,5) (1,9)
(2,14)
```

9 - Using Lamda expression check whether an input list is palindrome or not

palindrome :: Eq a => [a] -> String

palindrome list = (\x -> if x == reverse x then "Palindrome" else "Not a Palindrome") list

```
*Main> palindrome [1,2,1]
"Palindrome"
*Main> palindrome [1,1]
"Palindrome"
*Main> palindrome [1]
"Palindrome"
*Main> palindrome [1,2,3,4]
"Not a Palindrome"
*Main> palindrome "MADAM"
"Palindrome"
```

10 - Check whether each list in the list is palindrome or not

checkPal :: Eq a => [a] -> Bool

checkPal x = (\x -> x == reverse x) x

palindromeList :: Eq a => [[a]] -> [Bool]

palindromeList list = map checkPal list

```
*Main> palindromeList [[1,2,1],[1,2,3,4]]
[True,False]
*Main> palindromeList [[],[1,1],[1]]
[True,True,True]
*Main> palindromeList ["Abhi","Mam"]
[False,False]
*Main> palindromeList ["Abhi","MaM"]
[False,True]
```

Thankyou!!