**SET2**

**Roll number:cb.en.u4cse22527**

Q1)

isTriangular ::[[Int]]->String

isTriangular []="None"

isTriangular mat

|isUpper mat ="UPPER"++show(len mat)

|isLower mat ="Lower"++show(len mat)

|otherwise ="None"

len ::[[Int]]->Int

len []=0

len (x:xs)=1+len xs

isUpper :: [[Int]] -> Bool

isUpper mat = all (== 0) [mat !! i !! j | i <- [1..n-1], j <- [0..i-1]]

where n = len mat

isLower :: [[Int]] -> Bool

isLower mat = all (== 0) [mat !! i !! j | i <- [0..n-2], j <- [i+1..n-1]]

where n = len mat

main = do

print (isTriangular [])

print (isTriangular [[1,2,3], [0,4,5], [0,0,6]])

print (isTriangular [[1,0,0], [1,4,0], [1,4,6]])  


Q2)

splitAtElem :: Eq a => a -> [a] -> ([a], [a])

splitAtElem \_ [] = ([], [])

splitAtElem x (y:ys)

| x == y = ([], y:ys)

| otherwise = let (before, after) = splitAtElem x ys

in (y:before, after)

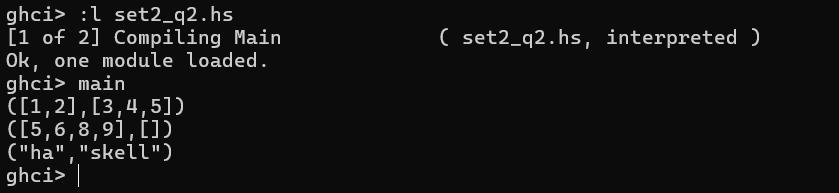
main :: IO ()

main = do

print $ splitAtElem 3 [1,2,3,4,5]

print $ splitAtElem 7 [5,6,8,9]

print $ splitAtElem 's' "haskell"



Q4)

sumGreaterThanN :: Int -> [Int] -> Int

sumGreaterThanN \_ [] = 0

sumGreaterThanN n (x:xs)

| x > n = x + sumGreaterThanN n xs

| otherwise = sumGreaterThanN n xs

sumGreaterThanNHO :: Int -> [Int] -> Int

sumGreaterThanNHO n lst = sum (filter (> n) lst)

main :: IO ()

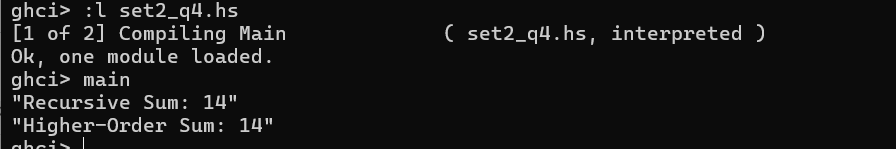
main = do

let lst = [3,5,1,6,8,4]

let n = 5

print $ "Recursive Sum: " ++ show (sumGreaterThanN n lst)

print $ "Higher-Order Sum: " ++ show (sumGreaterThanNHO n lst)



Q5)

temperatureDiff :: Double -> Double -> Double

temperatureDiff t1 t2 = abs (t1 - t2)

cumulativeTempDiff :: [Double] -> Double

cumulativeTempDiff temps = foldl1 (\acc t -> acc + abs t) (zipWith (-) temps (tail temps))

cumulativeTempDiffFoldr :: [Double] -> Double

cumulativeTempDiffFoldr temps = foldr (\t acc -> abs t + acc) 0 (zipWith (-) temps (tail temps))

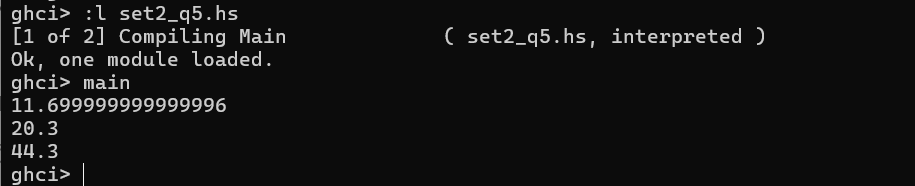
main :: IO ()

main = do

print $ temperatureDiff 32.6 44.3

print $ cumulativeTempDiff [32, 44, 38, 11.7]

print $ cumulativeTempDiffFoldr [32, 44, 38, 11.7]



Q6)

myFilterFunction :: (Int -> Bool) -> (Int -> Int) -> [Int] -> [Int]

myFilterFunction predicate func lst = [if predicate x then func x else x | x <- lst]

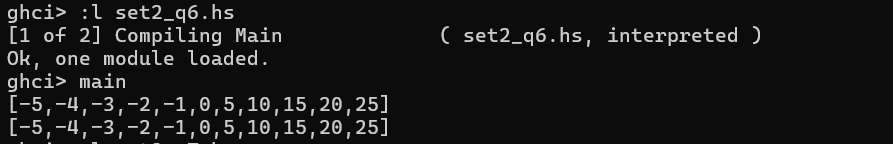
-- Test Cases

main :: IO ()

main = do

print $ myFilterFunction (>0) (\*5) [-5..5]

print $ myFilterFunction (>=0) (\*5) [-5..5]



Q7)

import Data.Char (isDigit)

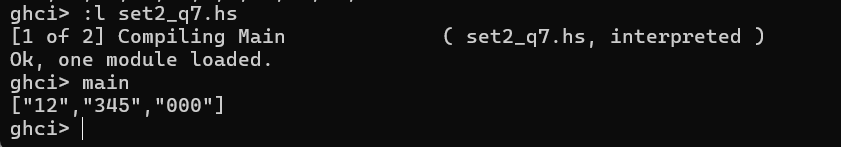
retainDigits :: [String] -> [String]

retainDigits lst = [filter isDigit str | str <- lst]

main :: IO ()

main = do

print $ retainDigits ["Hello12", "B3Y4E5", "000"]



Q8)

sortedmerge :: (Ord a, Eq a) => [a] -> [a] -> [a]

sortedmerge xs [] = xs

sortedmerge [] ys = ys

sortedmerge (x:xs) (y:ys)

| x < y = x : sortedmerge xs (y:ys)

| x == y = x : sortedmerge xs ys

| x > y = y : sortedmerge (x:xs) ys

-- Test Cases

main :: IO ()

main = do

print $ sortedmerge [1, 3, 5] [2, 3, 6]

print $ sortedmerge [1, 2, 2, 4] [2, 3, 4, 5]

print $ sortedmerge "abc" "bdf"

