1.

isVal :: [[Int]] -> String

isVal [] = "NONE"

isVal matrix

| isUpper && isLower = "NONE"

| isUpper = if isSize3 then "UPPER3" else "UPPER"

| isLower = if isSize3 then "LOWER3" else "LOWER"

| otherwise = "NONE"

where

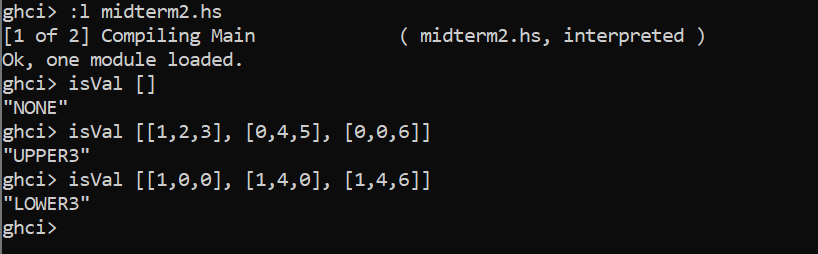
rowCount = length matrix

colCount = if null matrix then 0 else length (head matrix)

isUpper = all (\(i, row) -> all (== 0) (take i row)) (zip [0..] matrix)

isLower = all (\(i, row) -> all (== 0) (drop (i+1) row)) (zip [0..] matrix)

isSize3 = rowCount == 3 && colCount == 3



2.

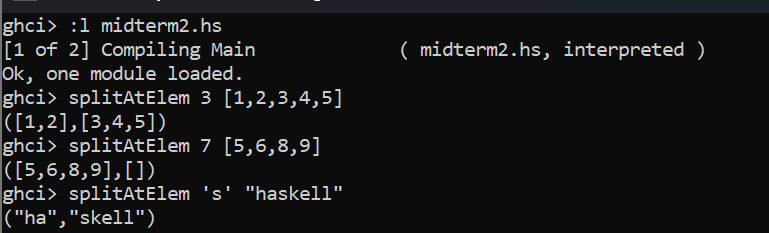
splitAtElem :: Eq a => a -> [a] -> ([a], [a])

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

splitAtElem x lst =

let (before, after) = break (== x) lst

in if null after then (before, []) else (before, after)



4.

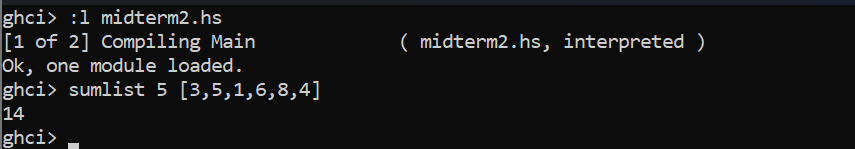
sumlist :: (Ord a,Num a) => a -> [a] -> a

sumlist n [] = 0

sumlist n (x:xs)

| x > n = x + sumlist n xs

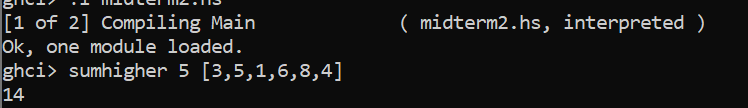
| otherwise = sumlist n xs



Higher Order Function approach

sumhigher :: (Ord a,Num a) => a -> [a] -> a

sumhigher n xs = sum (filter (>n) xs)

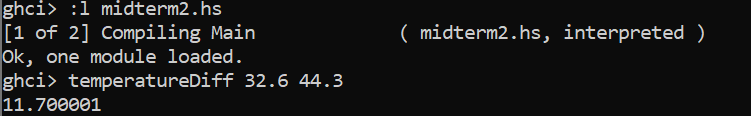


5.

a-

temperatureDiff :: Float -> (Float -> Float)

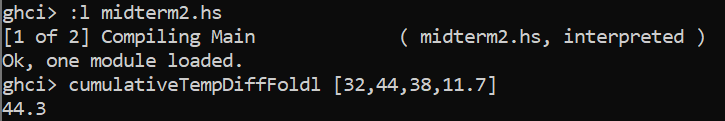
temperatureDiff t1 = \t2 -> abs (t1 - t2)



b-

cumulativeTempDiffFoldl :: [Double] -> Double

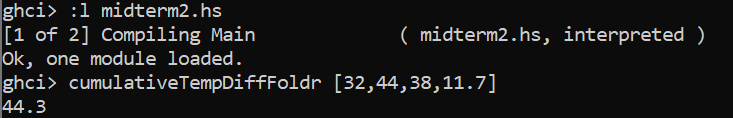
cumulativeTempDiffFoldl l = foldl1 (+) (zipWith (\x y -> abs (x - y)) l (tail l))



c-

cumulativeTempDiffFoldr :: [Double] -> Double

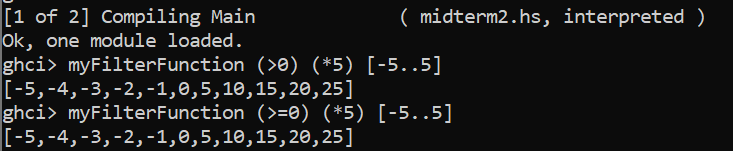
cumulativeTempDiffFoldr l = foldr1 (+) (zipWith (\x y -> abs (x - y)) l (tail l))



6.

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

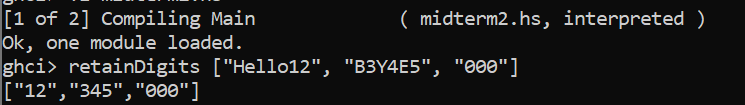
myFilterFunction pred func xs = [if pred x then func x else x | x <- xs]



7.

retainDigits :: [String] -> [String]

retainDigits = map (filter (`elem` ['0'..'9']))



8.

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

sortedmerge xs [] = xs

sortedmerge [] ys = ys

sortedmerge (x:a) (y:b)

| x < y = x : sortedmerge a (y:b)

| x == y = x : y : sortedmerge a b

| x > y = y : sortedmerge (x:a) b

