-- Zayadur Khan 02/10/17 CIS-252 M001 Assignment 4

import Data.Char

-- myProduct ns returns the product of all the numbers in ns

-- i.e. myProduct [3,6,2,10] returns 360

-- myproduct [7,18,2,0,9] returns 0

-- myProduct [] returns 1

myProduct :: [Integer] -> Integer

myProduct [] = 1

myProduct (n:ns) = n \* myProduct ns

-- shout str returns the string obtained by replacing all lowercase

-- letters in str by their uppercase equivalents; all other

-- characters remain unchanged

-- i.e. shout "Let's go, Orange!" returns

-- "LET'S GO, ORANGE!"

shout :: String -> String

shout "" = ""

shout (c:str) = toUpper(c) : shout str

-- zap c cs returns the string obtained by removing all occurences

-- of c from cs

-- i.e. zap 'a' "abbadaab" returns "bbdb"

zap :: Char -> String -> String

zap c "" = ""

zap c (e:cs)

| e /= c = e : (zap c cs)

| otherwise = zap c cs

-- pairUp xs returns the list obtained by pairing up the first two

-- elements of xs, then the third and fourth elements, etc;

-- if xs has an odd number of elements, the final element is

-- paired with itself

-- i.e. pairUp [3,5,2,9] returns [(3,5),(2,9)]

-- pairUp "abcde" returns

-- [(’a’,’b’),(’c’,’d’),(’e’,’e’)]

pairUp :: [a] -> [(a,a)]

pairUp [] = []

pairUp [a] = [(a,a)]

pairUp (w:x:xs) = (w,x) : pairUp xs

-- neighbors xs returns a list containing all the pairs of

-- neighboring elements from xs

-- i.e. neighbors [3,5,2,9] returns [(3,5),(5,2),(2,9)]

-- neighbors "abcde" returns

-- [(’a’,’b’),(’b’,’c’),(’c’,’d’),(’d’,’e’)]

neighbors :: [a] -> [(a,a)]

neighbors [] = []

neighbors [a] = [(a,a)]

neighbors (w:x:xs) = (w,x) : neighbors (x:xs)

-- bag represents data type structured as: (char, count)

-- where char is the character in the bag

-- and count is the frequency of char in the bag

bag1, bag2, bag3 :: [(Char,Int)]

bag1 = [('z',1), ('e',2), ('k',1)]

bag2 = [('y',2), ('a',1), ('n',1), ('c',1), ('e',1)]

bag3 = [('j',1), ('o',1), ('u',1), ('l',1), ('e',1)]

-- bagCount bag returns the total number of items in bag

-- i.e. bagCount bag1 returns 4

-- bagCount bag2 returns 6

-- bagCount bag3 returns 5

bagCount :: [(Char,Int)] -> Int

bagCount [] = 0

bagCount ((x,y):xs)

| y > 0 = y + bagCount xs

| otherwise = 0

-- addToBag ch bag returns the bag obtained by adding

-- one copy of ch to bag

-- i.e. addToBag 'y' bag1

-- returns [('z',1),('e',2),('k',1),('y',1)]

-- !!! NOT WORKING !!!

addToBag :: Char -> [(Char,Int)] -> [(Char,Int)]

addToBag ch ((x,y):xs)

| ch == x = [(x,y+1)] -- if ch already exists, add 1 to y

| otherwise = (ch,y) : addToBag ch xs -- else add data to list, run method again

-- removeFromBag ch bag returns the bag obtained by

-- removing one copy of ch from bag

-- i.e. removeFromBag 'e' bag1

-- returns [('z',1),('e',1),('k',1)]

-- !!! NOT WORKING !!!

removeFromBag :: Char -> [(Char,Int)] -> [(Char,Int)]

removeFromBag ch ((x,y):xs)

| ch == x = [(x,y-1)] -- if ch exists, remove 1 from y

-- | y < 1 = (drop (x,y)) -- if y is 0, drop the data

| otherwise = removeFromBag ch xs -- if ch doesn't exist, run again