CS344: Programming Languages
Homework 8: more on Haskell

Required Problems

1. Data compression is very useful because it helps reduce resources usage, such as data storage space or transmission capacity. For this task, you can assume that the input strings consist only of letters of the English alphabet.

   (a) Run-length encoding (RLE) is a very simple form of data compression in which runs of data are stored as a single data value and a count, rather than as the original run. Define a function rle that applied RLE to a the given string.

   ```
   rle :: String -> String
   rle "aaabbbbbc" ⟹ "3a 5b 1c"
   rle "banana" ⟹ "1b 1a 1n 1a 1n 1a"
   ```

   (b) Define rleInverse that applies the inverse RLE operation (RLE decoding) on a given string.

2. Define a `Point` as a type alias for a two-dimensional point with double precision (so `type Point = (Double, Double)`). Using `Point`, define `Polygon` as an alias for a list of points: each two adjacent points form a line segment on the boundary of the polygon. (Note that the first and last points will be considered adjacent, so we avoid repeated elements.) A polygon will only be valid if it has 3 or more points.

   (a) Define a function `dist` that calculates distance between two points:

   ```
   dist :: Point -> Point -> Double
   dist (0.5,3) (3.5,0) ) ⟹ 3
   dist (1.2,-1.8) (1.2,-1.8) ⟹ 0
   ```

   (b) Define `onLineSegment` that checks if point (the first argument) lies on the line segment (where the starting point is the second argument and the ending point is the third argument). Use 0.00001 precision. (Hint: use dist.)

   ```
   onLineSegment :: Point -> Point -> Point -> Bool
   onLineSegment (1,2) (0,0) (2,4) ) True
   onLineSegment (-2,-4) (0,0) (2,4) ) False
   ```

   (c) Define `isValid` that tests if the polygon is valid.

   ```
   isValid :: Polygon -> Bool
   isValid [] ⟹ False
   isValid [(0,0), (1.5,2)] ⟹ False
   isValid [(3.1,3), (3,3), (3,3)] ⟹ True
   ```

   (d) Define `perimeter` that returns the perimeter of a polygon. If the polygon is not valid, return an error message "Not a valid polygon".

   ```
   perimeter :: Polygon -> Double
   perimeter [(0,0), (0,1), (1,1), (1,0)] ⟹ 4
   perimeter [(0,0), (0,1)] ) ⟹ error "Not a valid polygon"
   ```

(e) Define `onPolygonBorder` that checks if the point is on a polygon border. If the polygon is not valid, return an error message “Not a valid polygon”. (Hint: use any or or, use onLineSegment.)

```
 onPolygonBorder ::  Point -> Polygon -> Bool
onPolygonBorder (1,2) [(0,0), (2,4), (0,6), (-5,0)] ) ⟹ True
onPolygonBorder (3,3) [(0,0), (2,4), (0,6), (-5,0)] ) ⟹ False
onPolygonBorder (3,3) [(3,3), (3,3), (3,3)] ) ⟹ True
onPolygonBorder (1,5) [(1,1)] ⟹ error "Not a valid polygon"
```

3. Extra credit: Write a function:

```
squash :: (a -> a -> a) -> [a] -> [b]
```

which applies a given function to adjacent elements in a list. For example, squash $f[x_1, x_2, x_3, x_4]$ should equal $[fx_1x_2, fx_2x_3, fx_3x_4]$.

You can implement this either using explicit recursion and pattern matching, or using the function `zipWith`. Or for a bit more extra credit, write two versions and solve it both ways.