Problem Sheet 3.1: Lists in Haskell

Maps

a) The function `isLowerSt` is similar to some of the fuctions we looked at last week: it converts all the letters in a string lower case, using the `Data.Char` function `toLower`. Rewrite `toLowerSt` using `map`.

```
*Main> toLowerSt "Hello World"
"hello world"
```

b) Complete the function `toLowerCons`, which only converts consonants to lower case. Using this `toLowerCons` and `map`, type and implement the function `toLowerConsSt`, which converts all the consonants in a string to lower case.

```
*Main> toLowerConsSt "HELLO WORLD"
"hEllO wOrld"
```

Filters

a) The function `onlyLetters` takes a string and only returns the letters (i.e. no digits or punctuation). It uses the `Data.Char` function `isLetter :: Char -> Bool`. Rewrite `onlyLetters` using `filter`.

```
*Main> onlyLetters "2 Fast 2 Furious"
"FastFurious"
```

b) Using the `Data.Char` functions `isDigit` and `isLetter`, and the `filter` function, type and implement the function `onlyNumsOrLetters`, which returns only the numbers and letters from an input string.

```
*Main> onlyNumsOrLetters "2 Fast 2 Furious"
"2Fast2Furious"
```

c) Sometimes, we will want to use both `map` and `filter` in the definition of a function. Both `onlyLettersToLower1` and `onlyLettersToLower2` should do the same thing: from an input string, convert all the letters to lowercase and erase all numbers and punctuation. However `onlyLettersToLower1` should erase non-letters first, then uncapitalise, whereas `onlyLetterstoLower2` should uncapitalise letters first and then erase non-letters. Type and implement both functions.

```
*Main> onlyLettersToLower1 "2 Fast 2 Furious"
"fastfurious"
*Main> onlyLettersToLower2 "2 Fast 2 Furious"
"fastfurious"
```

Zips

We have provided you with two lists: `firstnames` and `lastnames`. In these exercises, we will look at different ways of zipping them together.

a) We have given you the definition of `wholeNames`. Test it out to see what it does, work out what its type signature should be, then try to compile the file to see if you are correct.

b) countNames should take a list of names as an input, and output each name in a pair its position in the list. Type and implement `countNames`.

```
*Main> countNames firstNames
[(1,"Adam"),(2,"Brigitte"),(3,"Charlie"),(4,"Dora"),(5,"Engelbert")]
```

c) We have given you the definition of the function `wholeNames2`. Test it out to see what it does, then give it a type signature. Bonus: see if you can adapt `wholeNames2` to put a space in between the first and second names.

d) We have started the definition of the function `rollCall`. Complete the definition and type the function, so that the output when applied to the two given lists is like this. Feel free to customise your function a little.

```
*Main> rollCall firstNames
["1: Adam? 'Present!'","2: Brigitte? 'Present!'",
"3: Charlie? 'Present!'","4: Dora? 'Present!'",
"5: Engelbert? 'Present!'"]
*Main> rollCall secondNames
["1: Ashe? 'Present!'","2: Brown? 'Present!'",
"3: Cook? 'Present!'","4: De Santis? 'Present!'"]
```