Lab session 10 – Haskell

Unit	Programming languages: principles and design (6G6Z1110) Programming languages – SE frameworks (6G6Z1115)
Lecturer	Rob Frampton
Week	11

Description

The following exercises should be entered into a text editor and then executed under `ghc`. You can also do this in your web browser on *repl.it*.

If you prefer to work on your own laptop, or when working at home, you could install the Haskell compiler, which is available at https://www.haskell.org.

Exercises

Exercise 1

In this exercise you will complete the following code, which reverses the words in a string:

```
import Data.Char

-- Makes a String (an array of Char) lower case
toLowerString :: String -> String
toLowerString str = ??? toLower str

-- Joins an array of Strings together, separated by spaces
joinWithSpaces :: [String] -> String
joinWithSpaces lst = concat (map (\x -> ???) lst)

-- Splits the input string into separate words, returns array
reverseWords :: String -> String
reverseWords str = ???

myString = "This is a test string"

main = do
  putStrLn $ show $ reverseWords myString
```

There are three functions to complete:

- `toLowerString` transforms the input string by calling `toLower` on each element. You must provide the missing function to do this, in the space provided.
- `joinWithSpaces` adds a space to the end of each string in a list and concatenates them together. You must write an expression inside the anonymous function which adds a space (`" "`) to the end of the input, using the `++` operator.
- `reverseWords` takes a string and reverses the words.
    - Make the string lower case using the `toLowerString` function
    - Splits it into words using the `words` function
    - Reverses the resulting array using the `reverse` function
    - Joins the array back into a string using the `joinWithSpaces` function

Note: you may use the function composition notation shown in the second Haskell lecture if you like

When complete, your program should print: `string test a is this`

Exercise 2

In this exercise you will write a program which triples a list of numbers and then displays them as a bar chart.

a) Write a function called `getTriples` which takes a list as an argument and triples every element of the list. You can test your function by putting the following code in your file:

```
main = do
  putStrLn $ show $ getTriples [ 3, 1, 2, 4 ]
```

Which should give the following output:

```
[9,3,6,12]
```

Hint: You will need to use the `map` function, and pass it an anonymous function.

b) Write a function called `numberToXs` which takes an integer as an argument and returns a string containing a sequence of `"x"`'s (the number of `x`'s is given by the argument), followed by a newline. You can test your function by putting the following code in your file:

```
main = do
  putStrLn $ show $ numberToXs 5
```

Which should give the following output:

```
"xxxxx\n"
```

Hint: You will need to use the `replicate` function and the concatenation operator `++`.

c) Write a function called `listToXs` which takes a list of integers as an argument, and returns a string of `x`'s for each number, separated by newlines. You should use your `numberToXs` function from the previous part to generate this string. You can test your function by putting the following code in your file:

```
main = do
  putStrLn $ listToXs $ getTriples [ 3, 1, 2, 4 ]
```

Which should give the following output:

```
xxxxxxxxx
xxx
xxxxxx
xxxxxxxxxxxx
```

Hint: you will also need the `map` and `concat` functions.

Exercise 3

Write a function called `uniqueWords` which takes a string as an argument, and returns a list of the *unique* words in the string. You can test your function by putting the following code in your file:

```
main = do
  putStrLn $ show $ uniqueWords "one and two and one and three"
```

Which should give the following output:

```
["and","one","three","two"]
```

Hint: You will need to use at least the `group` and `sort` functions (see lecture slides) to group together unique words, after which you need to return a list containing each word once.

Exercise 4

In this exercise, you will write a function to give statistics about the number of vowels in a string.

a) Write a constant function called `vowels`, which returns a list of the five vowel characters `a`, `e`, `i`, `o` and `u`. You can test your function by putting the following code in your file:

```
main = do
  putStrLn $ show $ vowels
```

Which should give the following output:

```
"aeiou"
```

b) Write a function called `getLetterCount` which takes a character and a string as arguments, and returns the number of times the character appears in the string. You can test your function by putting the following code in your file:

```
main = do
  putStrLn $ show $ getLetterCount 'a' "many vowels in this sentence"
```

Which should give the following output:

```
1
```

Hint: You could use the `filter` and `length` functions, and you may need an anonymous function which compares input characters with the character parameter.

c) Write a function called `getLetterTuple`, which takes a character and a string as arguments, and returns a tuple in the form: (*character*, *number of times character appears in string*). You should use your `getLetterCount` function. You can test your function by putting the following code in your file:

```
main = do
  putStrLn $ show $ getLetterTuple 'a' "many vowels in this sentence"
```

Which should give the following output:

```
('a',1)
```

d) Write a function called `getVowelCount` which takes a string as an argument, and returns a list of tuples which describe the number of times each vowel appears in the input string. You should use your `getLetterTuple` function. You can test your function by putting the following code in your file:

```
main = do
  putStrLn $ show $ getVowelCount "many vowels in this sentence"
```

Which should give the following output:

```
[('a',1),('e',4),('i',2),('o',1),('u',0)]
```

e) Write a function called `getSortedVowelCount` which performs the same functionality as `getVowelCount`, but the list of tuples is sorted by their second element (i.e. the number of times the word appears). You can test your function by putting the following code in your file:

```
main = do
  putStrLn $ show $ getSortedVowelCount "many vowels in this sentence"
```

Which should give the following output:

```
[('e',4),('i',2),('o',1),('a',1),('u',0)]
```

Hint: see the lecture slides for an example of sorting tuples by their second element.