IN3043 Functional Programming
Exercises 4

1. This is a larger programming exercise, building a text-processing application from the list processing functions supplied by Haskell.

   You are to create a module `Histogram`, containing a function

   ```
   histogram :: String -> String
   ```

   that transforms an input string into a simple histogram showing the number of occurrences of each word in the original string.

   Specifically, if you type in GHCi (all on one line):

   ```
   putStr (histogram "Socrates: To be is to do. Sartre: To do is to be.
   Sinatra: Do be do be do.")
   ```

   you should get the output

   ```
         be ****
         do *****
         is **
     sartre *
    sinatra *
   socrates *
         to ****
   ```

   Notice that:

   - words are ordered alphabetically.
   - extra spaces have been added on the left (except for the longest word) so that the right-hand ends of the words line up. (Don't worry about oversize lines.)
   - all words are converted to lower case.
   - punctuation (except hyphens, not shown here) is removed.

   The `histogram` function should produce an output string contain: newline characters terminating the lines, i.e. the expression (again, all on one line):

   ```
   histogram "Socrates: To be is to do. Sartre: To do is to be.
   Sinatra: Do be do be do."
   ```

should produce a string with embedded newlines:

```
"      be ****\n      do *****\n      is **\n  sartre *\n sinatra *\ns
ocrates *\n      to ****\n"
```

`putStr` will be explained in Week 11, but it displays the string by acting on the newlines.

Try to design the implementation: Now think about how you're going to construct the required output string from the input. Break the problem up into pieces: What intermediate forms would be useful? How can you construct those?

Define auxiliary functions and test them separately. When you've got them doing what you want, define other functions using them. That is, design top-down and implement bottom-up, testing as you go.

If you're stuck, the next page has some hints at a possible design. (But don't look at them if you can design it yourself.)

Hints: Implement the function in stages:

(a) Write a function

```
lcwords :: String -> [String]
```

that takes a string and returns a list of lower-cased words without punctuation, e.g.

```
lcwords "Socrates: To be is to do. Sartre: To do is to be.
Sinatra: Do be do be do."
```

should return

```
["socrates","to","be","is","to","do","sartre","to","do","is","to"
,"be","sinatra","do","be","do","be","do"]
```

(b) Write a function

```
frequency :: [String] -> [(String, Int)]
```

that takes a list of words and produces a list of unique words, each paired with the number of times they occur in the original list, e.g.

```
frequency ["socrates","to","be","is","to","do","sartre","to","do",
"is","to","be","sinatra","do","be","do","be","do"]
```

should return

```
[("be",4),("do",5),("is",2),("sartre",1),("sinatra",1),("socrates"
,1),("to",4)]
```

(c) Write a function

```
formatLine :: Int -> String -> Int -> String
```

that takes the width of the left column, a word and a count, and returns a formatted histogram line, e.g.

```
formatLine 8 "be" 4
```

should produce the string

```
"      be ****"
```

(d) Write a definition of `histogram` that calls these functions. You'll be using the list of words and counts twice, so give it a name with a local definition (introduced with `where`).