Homework 1: Recollecting Basic Haskell
CIS 352: Programming Languages
22 January 2016, Version 1.2

Note: LYHGG = Miran Lipovača’s Learn You a Haskell for Great Good, see http://learnyouahaskell.com.

Background and Instructions

- This assignment is based on Chapter 1 of LYHGG.
- Use list comprehensions for these problems, NOT RECURSIONS.
- Use the file hw01.hs as a starting point for this assignment.
- For each problem, run the given QuickCheck tests for that problem. Also, add a few specific tests (non-QuickCheck) of your own.¹
- What to turn in: (i) your source code,² (ii) a transcript of your test runs, and (iii) the cover sheet.
- How to turn it in: See: http://www.cis.syr.edu/courses/cis352/reqs.html

Notes on quickCheck and testRun

QuickCheck is a Haskell debugging library we’ll be using a lot. For QuickCheck, a property is a Haskell function with a type of the form \( t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_n \rightarrow \text{Bool} \). If `convert_prop` is a property, then running

```haskell
globalError convert_prop
```

applies `convert_prop` to 100 random inputs. If the function returns True on all the inputs, quickCheck reports:

```
+++ OK, passed 100 tests.
```

If there was a failure (a False), quickCheck reports something like:

```
+++ Failed! Falsifiable (after 21 tests and 4 shrinks): 59
```

This means 59 failed the test and `convert` (the function being tested by `convert_prop`) has a problem you need to fix.³

The function `testRun` (defined in `hw01.hs`) runs all of the individual QuickCheck tests in `hw01.hs`. So when you have everything working, then evaluating `testRun` should result in:

```
E.g., test that (isVowel 'x') returns False and (isVowel 'u') returns True.
³ with your name in the comments SVP

¹ The (out-of-date and more-than-you-want-to-know) manual for QuickCheck version 1 can be found at http://www.cse.chalmers.se/~rjmh/QuickCheck/manual.html.
Your Problems

❖ Problem 1: Distance. ❖
Recall that the Euclidean distance between two points \((x_1, y_1)\) and \((x_2, y_2)\) in the plane is: \(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\). Suppose we have a type synonym

\[
\text{type Point} = (\text{Double}, \text{Double})
\]

Implement a Haskell function

\[
\text{distance :: Point} \rightarrow \text{Point} \rightarrow \text{Double}
\]
such that \((\text{distance pt1 pt2})\) returns the Euclidean distance between the two points.\(^4\)

Use QuickCheck with the property dist_prop to test this function.

❖ Problem 2: Testing for vowels. ❖
Implement a Haskell function

\[
\text{isVowel :: Char} \rightarrow \text{Bool}
\]
that tests whether a character is a lower-case vowel, i.e., one of: ‘a’, ‘e’, ‘i’, ‘o’, and ‘u’.\(^5\)

Use QuickCheck with vowel_prop to test this function.

❖ Problem 3: Disemvoweling. ❖
Implement a Haskell function

\[
\text{disemvowel :: String} \rightarrow \text{String}
\]
which, given a String value, returns that string with all the lowercase vowels removed. For example,

\[
\text{disemvowel "mississippi mud pie"}
\]
should return “msssspp md p”.

Use QuickCheck with disemv_prop to test this function.
Problem 4: Smash.
Implement a Haskell function

\[ \text{smash} :: \text{String} \to \text{String} \]

that takes a string \( s \) and returns the result of removing all non-letter characters from \( s \) and translating each uppercase letter to the corresponding lowercase letter. For example \( \text{smash "Fee, Fie, Foe, & Fum!!"} \) would return \"feefiefoefum\". Defining a helper function is perfectly OK.\(^6\) Note that in classical cryptography, a message is always smashed (to remove obvious clues) before being encrypted.

Use QuickCheck with \text{smash\_prop} to test this function.

Problem 5: Circular shift cyphers.
A circular shift cypher (with shift \( i \)) takes a plain text message \( m \) and

(i) smashes \( m \) and then

(ii) replaces each letter with the letter \( i \) places down in the alphabet.\(^7\)

E.g., a circular shift of \"Look, a zebra!!\" by 1 results in \"mpplbafcsb\". Also a shift of \"mpplbafcsb\" by -1 results in \"lookazebra\".

Implement a Haskell function

\[ \text{shift} :: \text{Int} \to \text{String} \to \text{String} \]

such that \( \text{shift} \ n \ s \) does a circular shift of \( n \)-places on the result of smashing \( s \). Use list comprehension and \text{toNum} and \text{toChar} defined in \text{hw01.hs}.

Use QuickCheck with \text{shift\_prop} to test this function.

Problem 6: Capitalization.
Implement a Haskell function

\[ \text{capitalized} :: \text{String} \to \text{String} \]

that takes a nonempty string and properly capitalizes it, i.e., the first character is upper case and the remaining characters are lower case. E.g., \( \text{capitalized "syRaCusE"} \) should return \"Syracuse\".

Use QuickCheck with \text{cap\_prop} to test this function.

Problem 7: Title Capitalization.
Implement a Haskell function

\[ \text{title} :: [\text{String}] \to [\text{String}] \]

that given a list of words, capitalizes them as a title. For this problem, that means

(i) each word over four characters long is capitalized, and

\(^6\) \text{N.B.} \text{isLetter} sadly doesn't do what you want since it is based on Unicode. However, \text{isLower} and \text{isUpper} behave as you'd expect.

\(^7\) When we run off the end of the alphabet, we wrap around from the front.

\textbf{Corrected:} Previously the type of \text{title} was incorrectly given as \([\text{String}] \to [\text{String}]\).
(ii) each word four or fewer characters in length is all lower case—except if it is the first word in the input list, in which case it is capitalized.

E.g., \texttt{(title ["the", "castle", "of", "wulfenbach"])}) should return ["The", "Castle", "of", "Wulfenbach"].

Use QuickCheck with \texttt{title\_prop} to test this function.

**Useful functions**

\[
\begin{array}{ll}
\&\&, \&|\& | & : \text{Bool} \to \text{Bool} \to \text{Bool} \\
(==), (/=) & : (\text{Eq a}) \Rightarrow a \to a \to \text{Bool} \\
(++) & : (\text{Floating a}) \Rightarrow a \to a \to a \\
(++) & : a \to [a] \to [a] \\
abs & : (\text{Num a}) \Rightarrow a \to a \\
chr & : \text{Int} \to \text{Char} \\
div, \text{mod} & : (\text{Integral a}) \Rightarrow a \to a \to a \\
divMod & : (\text{Integral a}) \Rightarrow a \to a \to (a,a) \\
elem, \text{notElem} & : \text{Eq a} \Rightarrow a \to [a] \to \text{Bool} \\
head, \text{last} & : [a] \to a \\
init, \text{tail} & : [a] \to [a] \\
isLetter, isLower, isUpper & : \text{Char} \to \text{Bool} \\
length & : [a] \to \text{Int} \\
not & : \text{Bool} \to \text{Bool} \\
ord & : \text{Char} \to \text{Int} \\
maximum, minimum & : (\text{Ord a}) \Rightarrow [a] \to a \\
product, \text{sum} & : (\text{Num a}) \Rightarrow [a] \to a \\
sqrt & : (\text{Floating a}) \Rightarrow a \to a \\
toLower, toUpper & : \text{Char} \to \text{Char}
\end{array}
\]