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. E.g., test that (isVowel 'x') returns False and (isVowel 'u') returns True.
- What to turn in: (i) your source code (with your name in the comments SVP), (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

Grading criteria

- The homework is out of 100 points.
- Each problem is worth 14 points (= 10 pts correctness + 4 pts testing).
- You get 2 points for putting your name in the source code file.

Notes on quickCheck and testRun

QuickCheck is a Haskell debugging library we’ll be using a lot in this course. For QuickCheck, a property is a Haskell function with a type of the form t1 -> t2 -> · · · -> tn -> Bool. If convert_prop is a property, then running

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 The (out-of-date and more-than-you-want-to-know) manual for QuickCheck version 1 can be found at http://www.cse.chalmers.se/~rjmh/QuickCheck/manual.html.

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:

*Main> testRun
convert prop : +++ OK, passed 100 tests.
vowel prop : +++ OK, passed 100 tests.
disemvowel prop : +++ OK, passed 100 tests.
smash prop : +++ OK, passed 100 tests.
shift prop 1 : +++ OK, passed 100 tests.
shift prop 2 : +++ OK, passed 100 tests.
capitalized prop : +++ OK, passed 100 tests.
title prop : +++ OK, passed 100 tests.

If the above isn’t the result, you have more work to do.
Problems

Problem 1: Time conversion.

Implement a Haskell function

\[\text{convert} :: \text{Int} \rightarrow (\text{Int}, \text{Int}, \text{Int})\]

The function takes a nonnegative \text{Int} \( t \) as input (the number of seconds since midnight) and returns a triple of nonnegative nonnegative \text{Int}s \((h, m, s)\) where these give the hours, minutes, and seconds since midnight. (I.e., \( t = 3600 \times h + 60 \times m + s \) where \( h \geq 0 \) and \( 60 > m, s \geq 0 \).)

Use QuickCheck with the property convert_prop to test this function.

Problem 2: Testing for vowels.

Implement a Haskell function

\[\text{isVowel} :: \text{Char} \rightarrow \text{Bool}\]

that tests whether a character is a lower-case vowel, i.e., one of: 'a', 'e', 'i', 'o', and 'u'. (Hint: Use elem.)

Use QuickCheck with the property vowel_prop to test this function.

Problem 3: Disemvoweling.

Implement a Haskell function

\[\text{disemvowel} :: \text{String} \rightarrow \text{String}\]

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

\[\text{disemvowel} \text{ "mississippi mud pie"}\]

should return \text{"msssspp md p"}.

Use QuickCheck with the property disemv_prop to test this function.

Problem 4: Smash.

Implement a Haskell function

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

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 (smash "Fee, Fie, Foe, & Fum!!") would return "feefiefoefum". Defining a helper function is perfectly OK. (N.B. isLetter sadly doesn’t do what you want since it is based on Unicode. However, isLower and isUpper behave as you’d expect.) Note that in classical cryptography, a message is always smashed (to remove obvious clues) before being encrypted.

Use QuickCheck with the property 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. (When we run off the end of the alphabet, we wrap around from the front.)

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

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

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

Use QuickCheck with the property shift_prop to test this function.
**Problem 6: Capitalization.**
Implement a Haskell function

```haskell
capitalized :: String -> 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.,

(capitalized "syRaCusE") should return "Syracuse".

Use QuickCheck with the property `cap_prop` to test this function.

**Problem 7: Title Capitalization.**
Implement a Haskell function

```haskell
title :: [[String]] -> [[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
(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., (title ["the", "castle", "of", "wulfenbach"]) should return ["The", "Castle", "of", "Wulfenbach"].

Use QuickCheck with the property `title_prop` to test this function.

---

**Useful functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(&amp;&amp;)</code>, `(</td>
<td></td>
</tr>
<tr>
<td><code>(==)</code>, <code>/=</code></td>
<td><code>(Eq a) =&gt; a -&gt; a -&gt; Bool</code></td>
</tr>
<tr>
<td>`(**)</td>
<td><code>(Floating a) =&gt; a -&gt; a -&gt; a</code></td>
</tr>
<tr>
<td><code>(:)</code></td>
<td><code>a -&gt; [a] -&gt; [a]</code></td>
</tr>
<tr>
<td><code>(++)</code></td>
<td><code>[a] -&gt; [a] -&gt; [a]</code></td>
</tr>
<tr>
<td><code>abs</code></td>
<td><code>(Num a) =&gt; a -&gt; a</code></td>
</tr>
<tr>
<td><code>chr</code></td>
<td><code>Int -&gt; Char</code></td>
</tr>
<tr>
<td><code>div</code>, <code>mod</code></td>
<td><code>(Integral a) =&gt; a -&gt; a -&gt; a</code></td>
</tr>
<tr>
<td><code>divMod</code></td>
<td><code>(Integral a) =&gt; a -&gt; a -&gt; (a,a)</code></td>
</tr>
<tr>
<td><code>elem</code>, <code>notElem</code></td>
<td><code>Eq a =&gt; a -&gt; [a] -&gt; Bool</code></td>
</tr>
<tr>
<td><code>head</code>, <code>last</code></td>
<td><code>[a] -&gt; a</code></td>
</tr>
<tr>
<td><code>init</code>, <code>tail</code></td>
<td><code>[a] -&gt; [a]</code></td>
</tr>
<tr>
<td><code>isLetter</code>, <code>isLower</code>, <code>isUpper</code></td>
<td><code>Char -&gt; Bool</code></td>
</tr>
<tr>
<td><code>length</code></td>
<td><code>[a] -&gt; Int</code></td>
</tr>
<tr>
<td><code>not</code></td>
<td><code>Bool -&gt; Bool</code></td>
</tr>
<tr>
<td><code>ord</code></td>
<td><code>Char -&gt; Int</code></td>
</tr>
<tr>
<td><code>maximum</code>, <code>minimum</code></td>
<td><code>(Ord a) =&gt; [a] -&gt; a</code></td>
</tr>
<tr>
<td><code>product</code>, <code>sum</code></td>
<td><code>(Num a) =&gt; [a] -&gt; a</code></td>
</tr>
<tr>
<td><code>sqrt</code></td>
<td><code>(Floating a) =&gt; a -&gt; a</code></td>
</tr>
<tr>
<td><code>toLower</code>, <code>toUpper</code></td>
<td><code>Char -&gt; Char</code></td>
</tr>
</tbody>
</table>