# Haskell Review 3

### Bubblesort

The *bubblesort* algorithm works by exchanging adjacent elements that are out of order until all the elements are in the right order.

1. Write a function `bubble :: Ord a => [a] -> [a]` which recursively goes through the list and interchanges every element that is followed by a smaller element.

   ```
   bubble [1,3,2] = [1,2,3]
   ```

   ```
   bubble [1,3,5,4,3,2] = [1,3,4,3,2,5]
   ```

2. Write a function `bubbleSort :: Ord a => [a] -> [a]` which repeatedly applies the `bubble` operator to the given list until it has no effect. (So that calling `bubble` results in the same list.)

### Searching and replacing substrings

If $u$ is a string, then $v$ is a *substring* if $u$ can be written as $u = xvy$ for some strings $x$ and $y$. The strings $x$ and $y$ can be empty. In particular, `ba` is a substring of `abcba`, as is `abcba`. Note also that the empty string is a substring of every string.

1. Write a function `findString :: String -> String -> Bool` which returns true if the first argument is a substring of the second

   ```
   findString "aba" "abcba" = False
   ```

   ```
   findString "bc" "abcba" = True
   ```

   *Hint.* You may use a helper function `isPrefix` defined in the previous homework.

2. Write a function `genSubstrings :: String -> [String]` which generates all the substrings of a given string and puts them in a list.

   ```
   genSubstrings "abcd" = ["","a","ab","b","abc","bc","c","abcd","bcd","cd","d"]
   ```

   You may list the substrings in a different order, but your list should avoid any duplicate entries.

   *Hint.* Write a helper function to generate all prefixes of a list, then use it recursively.

3. Write a function `replacePrefix :: (String,String) -> String -> String` which takes a pair `(old,new)` of strings, another string `str`, and replaces the prefix `old` of `str` with the string `new`.

   The function can have undefined behavior if `old` is not a prefix of `str`.

   ```
   replacePrefix ("be","spe") "bear" = "spear"
   ```

   *Hint.* There is a simple solution to this question using the `drop` function.

4. Write a function `replaceString :: (String,String) -> String -> String` which replaces the first occurrence of a substring with a new string:

   ```
   replaceString ("morning","evening") "Good morning!" = "Good evening!"
   ```

### A simple cypher

A *substitution cypher* is a simple encoding technique where each character in the input text is mapped to another character according to some fixed permutation of the alphabet.

We can represent such a permutation by a lookup table — a list of key–value pairs.

1. Write a function `lookUp :: Char -> [(Char,Char)] -> Char` so that calling `lookUp x perm` searches the list `perm` for the pair `(x,y)` whose first coordinate is the given character `x`. It should then output the second coordinate `y`.

   ```
   lookUp 'C' [('A','X'),('B','Y'),('C','Z'),('D','W')] = 'Z'
   ```

2. Write a function `encode :: [(Char,Char)] -> String -> String` which takes a lookup table and a string and replaces every character by the value it is mapped to by the given table.

   ```
   encode [('A','X'),('B','Y'),('C','Z'),('D','W')] "BABA" = "YXYX"
   ```

   (How could you implement the decode function?)

3. Write a function `makeTable :: String -> String -> [(Char,Char)]` which takes two strings and creates a table by pairing up their characters at the same position.

   ```
   makeTable "ABCD" "XYZW" =  [('A','X'),('B','Y'),('C','Z'),('D','W')]
   ```

Once you implemented the functions above, you can make a Caesar cypher by rotating the English alphabet by a fixed number of letters:

```
caesar :: Int -> [(Char,Char)]|
caesar n = makeTable abc (drop n (cycle abc)) where abc = ['A'..'Z']
```

## Submission instructions

- Save your file as `LASTNAMErev3.hs`. Submit it to asulearn by the deadline.
- *Your file must compile/load correctly.*

  If your file does not load, gives a type error or some formatting error, you will not receive any credit for this assignment.

- *You should provide a definition for every function in the assignment.*

  If you can't get a function to work, use the `error` construct to complete the answer.

  ```
  encode :: [(Char,Char)] -> String -> String
  encode x y = error "not done"
  ```