# Programming in Haskell – Homework Assignment 2

UNIZG FER, 2014/2015

Handed out: October 12, 2014. Due: October 19, 2014 at 23:59

*Note:* Define each function with the exact name specified. You can (and in most cases you should) define each function using a number of simpler functions. Unless said otherwise, a function may not cause runtime errors and must be defined for all of its input values. Use the `error` function for cases in which a function should terminate with an error message. Problems marked with a star ($\star$) are optional.

1. (a) Define a function `toTitleCase` that takes a `String` and returns a titlecased version of it. Articles, conjunctions, and prepositions are to be capitalized as well.

   ```
   toTitleCase "this is an example" ⇒ "This Is An Example"
   toTitleCase "100 bottles of beer on the wall"
   ⇒ "100 Bottles Of Beer On The Wall"
   toTitleCase "short" ⇒ "Short"
   toTitleCase "" ⇒ ""
   ```

   (b) In title case, articles, conjunctions, and prepositions are usually not capitalized, unlike in problem (a). To achieve this, we can define the function `toTitleCase'` that takes an additional argument: a list of words that will remain in lowercase, unless the word occurs at the beginning of the string, in which case it should be titlecased as well.

   ```
   toTitleCase' "Scientists discover a cure" ["a","the"]
   ⇒ "Scientists Discover a Cure"
   toTitleCase' "A day in the life" ["a","in","the"]
   ⇒ "A Day in the Life"
   ```

2. Implement a function `trimN` that takes a list and a number `n`. It removes `n` elements from each side of the list. If the trimming would eliminate more elements than the list contains, the list is returned unchanged.

   ```
   trimN [1,2,3,4,5] 2 ⇒ [3]
   trimN [] 5 ⇒ []
   trimN [1,2,3] 2 ⇒ [1,2,3]
   ```

3. Import `System.Environment` to get access to the `getArgs` action, which returns a list of command line arguments to the program. Given a single path as an argument, read the file at the given path and print it out with all of its characters capitalized. To read in a file, use the `readFile` IO action.

   ```
   $ ./ECHOFILE file.txt
   THIS IS SOME TEXT IN FILE.TXT
   ```

Here is an example of using `getArgs` to print out the first command line argument:

```
import System.Environment
printFirstArg = do
 args <- getArgs
 putStrLn $ head args
```

4. Define a function `onlyDivisible` that takes a `String` and a number `n`. It should drop all characters located at positions that are not divisible by `n`. Character positions are 0–indexed.

   ```
   onlyDivisible "example" 1 ⇒ "example"
   onlyDivisible "11+111=122" 2 ⇒ "1+1=2"
   onlyDivisible "This is the third example case" 10 ⇒ "Tea"
   onlyDivisible any 0 ⇒ error "n must be positive!"
   ```

5. You are given a set of points in the Cartesian coordinate system, represented as a list of tuples. Implement a function `triangleCounter` that returns the number of distinct triangles with endpoints from the given set. Degenerate triangles (formed using three colinear points) should not be included.

   ```
   triangleCounter [] ⇒ 0
   triangleCounter [(0,0),(1,1),(2,2)] ⇒ 0
   triangleCounter [(0,0),(1,1),(0,2)] ⇒ 1
   triangleCounter [(0,0),(0,1),(1,0),(1,1)] ⇒ 4
   ```

6. Implement a function `reverseWords` that takes a sentence and reverses the order of words. A sentence is represented as a `String` with words separated by a single whitespace.

   ```
   reverseWords "" ⇒ ""
   reverseWords "human being and fish can coexist peacefully"
   ⇒ "peacefully coexist can fish and being human"
   ```

7. Let's use lists as sets, even though this isn't recommended for performance reasons. Define `intersect'` and `difference` that implement the usual set operators. Assume the list elements don't have an ordering (i.e., don't use operators such as `<` or `>` to compare them, and also don't sort them), but do assume we can test them for equality. Both input and output lists may have duplicate elements.

   ```
   intersect' "mio" "mao" ⇒ "mo"
   intersect' [1..3] [1..] ⇒ [1..3]
   intersect' [1..]  [1..3] ⇒ [1,2,3,⊥]
   intersect' [] [1..] ⇒ []
   intersect' [1..]  [] ⇒ []
   ```

   ```
   difference "mio" "mao" ⇒ "i"
   difference [4,3] [1..] ⇒ []
   difference [4,3] [1,3,2,12] ⇒ [4]
   difference [4,3] [5..] ⇒ [⊥]
   difference [] [1..] ⇒ []
   difference [1..]  [] ⇒ [1..]
   ```

8. We can use a list of lists, where all sublists are of equal length, to represent a matrix. Define the following functions over such a matrix:

(a) The function `isWellFormed` that checks whether the matrix has all rows of equal length.

```
isWellFormed [[1,2,3],[4,5,6],[7,8,9]] ⇒ True
isWellFormed [[1,2,3],[4,5]] ⇒ False
isWellFormed [[]] ⇒ False
```

(b) The function `size` that returns the dimensions of a $n \times m$ matrix as a tuple `(n,m)`.

```
size [[1,2,3],[4,5,6]] ⇒ (2,3)
size [[5,0,5],[2]] ⇒ error "Matrix is malformed"
size [[]] ⇒ error "Matrix is malformed"
```

(c) The function `getElement` that returns the element at the given position in matrix.

```
getElement [[9,8,7],[6,5,4],[3,2,1]] 0 0 ⇒ 9
getElement [[9,8,7],[6,5,4],[3,2,1]] 3 0
⇒ error "Index out of bounds"
getElement [[3,2,1],[5,4]] 0 0 ⇒ error "Matrix is malformed"
getElement [[1],[2,3]] (-1) 0 ⇒ either of the errors above
```

(d) The function `getRow` that returns the `i`–th row of a matrix.

```
getRow [[1,2],[3,4],[5,6]] 0 ⇒ [1,2]
getRow [[1,2,3],[4,5]] 1 ⇒ error "Matrix is malformed"
getRow [[1,2,3]] (-3) ⇒ error "Index out of bounds"
getRow [[1,2],[3,4,5]] 2 ⇒ either of the errors above
```

(e) The function `getCol` that returns the `i`–th column of a matrix.

```
getCol [[1,2,3],[4,5,6]] 0 ⇒ [1,4]
getCol [[1,2,3],[4,5]] 1 ⇒ error "Matrix is malformed"
getCol [[1,2,3]] (-3) ⇒ error "Index out of bounds"
getCol [[1,2],[3,4,5]] 2 ⇒ either of the errors above
```

(f) The function `addMatrices` that returns the sum of two given matrices.

```
addMatrices [[1,2,3],[4,5,6]] [[7,8,9],[10,11,12]]
⇒ [[8,10,12],[14,16,18]]
addMatrices [[1,2,3],[4,5,6]] [[1]]
⇒ error "Matrices are not of equal size"
addMatrix [[1,2],[3,4]] [[5,6],[7]] ⇒ error "Matrix is malformed"
```

(g) The function `transpose'` that returns a transposed version of the given matrix. It may not use the `Data.List.transpose` function.

```
transpose' [[1,2,3],[4,5,6]] ⇒ [[1,4],[2,5],[3,6]]
transpose' [[1,2,3],[4,5]] ⇒ error "Matrix is malformed"
```

(h)⋆ The function `multMatrices` that multiplies two matrices.

```
multMatrices [[1,2],[3,4],[5,6]] [[7,8],[9,10]]
⇒ [[25,28],[57,64],[89,100]]
multMatrices [[1,0],[0,1]] [[3,5],[9.2]] ⇒ [[3,5],[9,2]]
multMatrices [[1,2],[3,4]] [[5]]
⇒ error "Incompatible matrix dimensions"
multMatrices [[1,2],[3]] [[4]] ⇒ error "Matrix is malformed"
```