Programming in Haskell – Homework Assignment 2

UNIZG FER, 2015/2016

Handed out: October 23, 2014. Due: October 29, 2015 at 17:00

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.

Each problem is worth a certain number of points. The points are given at the beginning of each problem or subproblem (if they are scored independently). These points are scaled, together with a score for the in-class exercises, if any, to 10. Problems marked with a star ($\star$) are scored on top of the mandatory problems, before scaling. The score is capped at 10, but this allows for a perfect score even with some problems remaining unsolved.

1. *(3 pts)* 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 ::  [[Int]] -> Bool

   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 pair `(n,m)`.

   ```
   size ::  [[Int]] -> (Int, Int)

   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 ::  [[Int]] -> Int -> Int -> Int

   getElement [[9,8,7],[6,5,4],[3,2,1]] 1 0 ⇒ 6
   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 ::  [[Int]] -> Int -> [Int]

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 ::  [[Int]] -> Int -> [Int]

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 ::  [[Int]] -> [[Int]] -> [[Int]]

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' ::  [[Int]] -> [[Int]]

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 ::  [[Int]] -> [[Int]] -> [[Int]]

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"
```

2. *(3 pts)* Your task is to implement simple versions of the following unix command-line utilities. The main part of each utility should be inside a pure function and at the end they will be coupled with input/output.

   (a) The `wc` (wordcount) utility, printing the number of lines, words and characters within the string.

   ```
   wc ::  String -> (Int, Int, Int)

   let test = "Tiny sample file\ncontaining a haiku\nvery impressive"
   wc test ⇒ (3 8 51)
   ```

   (b) The `paste` utility, pairing up lines from two inputs and joining them with a tab character.

   ```
   paste ::  [String] -> [String] -> [String]

   let ps = ["not", "is not", "at all, really."]
   paste (lines test) ps
   ⇒ ["Tiny sample file\tnot, "containing a haiku\tis not",
   "very impressive\tat all, really."]

   let moreLines = ["This one, "has more, "lines"]
   let lessLines = ["no"]
   paste moreLines lessLines
   ⇒ ["This one\tno", "has more", "lines"]
   ```

   (c) The `cut` utility, taking a delimiter, index and contents as lines, then cuts out a portion of each line. The parts are separated by the given delimiter and the output part is determined by the given index. Column numbering starts from 1. You can assume separators will always be of length 1.

   ```
   cut ::  String -> Int -> [String] -> [String]

   let scores = ["1#Marko#99.85", "2#Iva#99.30", "3#Pasko#90.00"]
   cut "#" 2 scores
   ⇒ ["Marko", "Iva", "Pasko"]
   ```

   (d) Let's make the code runnable from the command line and define the `main` function. It will take arguments on the command line, with the first being the name of our utility function.

   Hint: Take a look at `readFile` and `System.Environment.getArgs`.

   ```
   > ./Homework2 wc test.txt
   3 8 51

   > ./Homework2 paste test.txt ps.txt
   Tiny sample file    not
   containing a haiku    is not
   very impressive   at all, really.
   ```

```
> ./Homework2 cut # 2 scores.txt
Marko
Iva
Pasko
```

3. *(3 pts)* You are given an undirected graph G defined by the number of nodes n and a list of edges e represented as a list of pairs. More formally, if a tuple (u, v) is an element of list e, then there exists an edge between node u and node v. Your job is to define a function `cyclicWalks` that finds the number of cyclic walks of length k in graph G.

   Hint: Number of distinct cycles of length k from node i corresponds to $(A_g^k)_{i,i}$ where A_g represents the adjacency matrix of graph G. Also, some functions from first task might be helpful.

   Helper: If you need help calculating A_g^k, the helper function below may be of use. It takes a function, the number of times to apply it and the argument to apply it to.

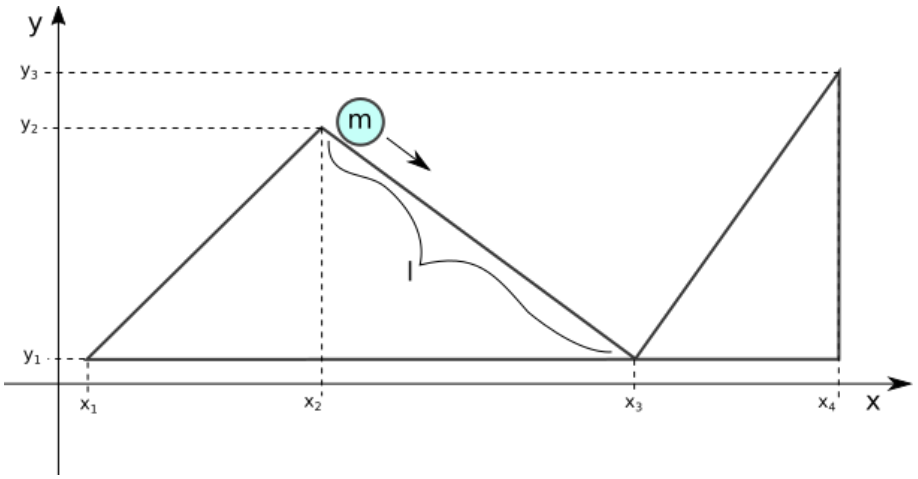
```
applyN :: (a -> a) -> Int -> a -> a
applyN f n = foldr (.) id (replicate n f)

type Node = Int
type Edge = (Node, Node)
cyclicWalks :: [Edge] -> Int -> Int

cyclicWalks [(1, 2), (3, 4), (2, 3), (1, 3)] 3
⇒ 6
cyclicWalks [(1, 2), (3, 4), (2, 3), (1, 3)] 4
⇒ 28
cyclicWalks [(1, 2), (3, 4), (2, 3), (1, 3)] 2
⇒ 8
cyclicWalks [(1, 2), (3, 4), (2, 3), (1, 3)] 0
⇒ error "Cycle length must be greater than 1"
```

4. (⋆) *(3 pts)* This task is loosely based on a lab assignment one could perform while studying classical mechanics. More precisely, we are trying to determine the minimum initial speed needed for a $1kg$ ball to pass through an obstacle course formed by a series of connected slopes.

   The coordinates of slopes' endpoints are represented by a list of tuples (by increasing x coordinate). Note that, for instance, the second element of the list represents both the end of the first slope and the beginning of the second slope.



When the ball of mass m rolls over a slope of length l you may assume that its speed will increase/decrease by $m * l$. The ball passes the course if its speed after each slope is greater than or equal to zero.

```
type Point = (Double, Double)
minSpeed :: [Point] -> Double

minSpeed [(0, 0), (1, 1), (2, 0)] ⇒ 1.414
minSpeed [(0, 0), (1, 1), (2, 0), (4, 3), (5, 0)] ⇒ 3.606
```

Corrections

Revision 1.1 – Fix the `getElement` function signature and make arguments clearer by changing example.
Revision 1.2 – Fix the erroneous sentence construction in task 3.
Revision 1.3 – Fix the second example in task 3.
Revision 1.4 – Fix a typo in `IO wc` example in task 2.
Revision 1.5 – Fix a typo in `paste` example in task 2.
Revision 1.6 – Rename `cyclicPaths` to `cyclicWalks`, fix the examples and add a helper function to avoid having to write your own recursion. *Note* if you have already named your function `cyclicPaths` and have submitted your HA, we will accept your submission.