4. Functional Programming / Haskell (25 Marks)

Consider writing a Domain-specific language for matrix manipulation. We use the following data type to represent any matrix of floating-point numbers:

```
data Matrix = Row [Float] Matrix | Null
```

That is, a matrix is represented as a sequence of rows given as lists of floating-point numbers. So, for example, the matrix

$$\begin{pmatrix} 2.3 & 4.5 & 1.2 \\ -0.4 & 3.2 & 3.4 \end{pmatrix}$$

is represented in the following way:-

```
m1 :: Matrix
m1 = Row [2.3, 4.5, 1.2] (Row [-0.4, 3.2, 3.4] Null)
```

Notice that the `Null` constructor just marks the end of the rows in a matrix.

In this question you may use standard Prelude and IO type, but no other Haskell libraries. Ensure that type definitions are given for all functions you write.

(a) Represent the following matrix using the data type `Matrix` and assign to variable `m2`. **[2 marks]**

$$\begin{pmatrix} 1.2 & -1.0 \\ 4.5 & -0.9 \\ 2.3 & 1.8 \end{pmatrix}$$

(b) We will say that a matrix is legitimate if and only if all rows have exactly the same length. Write a function `legitMatrix :: Matrix -> Bool` to check if a matrix is legitimate, returning `True` if it is, and `False` otherwise. **[3 marks]**

(c) Write a function `sizeMatrix` to calculate the size of a matrix assuming it is legitimate, returning a pair (x,y) where x is number of rows and y is number of columns. **[2 marks]**

(d) Write a function `insertColumn` to insert a list of floating-point numbers as a column to the left of an existing matrix.

For example, `insertColumn m1 [8.7, -2.5]` will return the matrix

$$\begin{pmatrix} 8.7 & 2.3 & 4.5 & 1.2 \\ -2.5 & -0.4 & 3.2 & 3.4 \end{pmatrix}$$

[2 marks]

(e) Consider this function:

```
emptyMatrix :: Int -> Matrix
emptyMatrix 0 = Null
emptyMatrix n = Row [] (emptyMatrix (n-1))
```

What does `emptyMatrix 3` return? **[1 mark]**

(f) Write a function `transposeMatrix` to implement the transpose of a matrix. **[3 marks]**

Hint: Use `insertColumn` and `emptyMatrix` functions for this question part.
Remember: the transpose of a matrix swaps rows for columns. For example, the transpose of `m1` will be

$$\begin{pmatrix} 2.3 & -0.4 \\ 4.5 & 3.2 \\ 1.2 & 3.4 \end{pmatrix}$$

(g) Write a function `addRows` to add together consecutive elements of two rows to return a new row, using list comprehension and the `zip` function, assuming they are the same length.

For example, `addRows [3.4, 6.7, 1.2] [-0.5, 1.0, -1.2]` will return `[2.9, 7.7, 0.0]`. **[3 marks]**

(h) Write an operator `+#+` to add two matrices together, using normal matrix addition.
You may use the `addRows` function to do this. **[2 marks]**

(i) We know that matrix addition is mathematically only correct if the two matrices being added are the same size. Write a safe operator `+##+` to add two matrices together, but using the `Maybe` data type, so that it returns `Nothing` in the case that either of the two matrices not being legitimate (see part (b) above) or they are not the same size. **[3 marks]**

Hint: You may use the `legitMatrix` and `sizeMatrix` functions and `+#+` operator already defined.

(j) Use the IO type to write a function `showMatrix` to display a matrix in the following way, with tabs between columns. For example,

```
showMatrix m1
```

writes to output:

```
[2.3    4.5     1.2]
[-0.4   3.2     3.4]
```

```
showMatrix m2
```

writes to output:

```
[1.2    -1.0]
[4.5    -0.9]
[2.3    1.8]
```

[4 marks]

Hint 1: You may use one or more auxiliary functions.
Hint 2: `show n` converts a number `n` to `String`, and `'\t'` is a tab character.