COMP 141: Haskell — Part 5

Instructions: In this exercise, we are going to review a bunch of Haskell structures.

(1) Consider the following function.

```
splitBy3 xs = [take ((length xs) `div` 3) xs ,
               drop ((length xs) `div` 3) xs ]
```

Run it on a few inputs to realize what it does.

(a) Redefine this function using `where` binding (in order to make it look nicer).

(b) Try the same with `let` binding.

(2) Use case expressions to define function `multList2` that receives a list of numbers and returns a list in which every element of the input list is multiplied by 2. Do not forget to annotate the types.

(3) Define your own version of function `drop`. Call it `drop'`. Do not forget to annotate appropriate types.

(4) Define your own version of function `cycle`. Call it `cycle'`. Do not forget to annotate appropriate types.

(5) Define function `splitPair :: [(a,b)] -> ([a],[b])` that receives a list of pairs and decomposes it into a pair of list of items, where the first list consists of the first component of each pair, and the second list consists of the second component of each pair. Use `where` binding for defining the function.

Example: `splitPair [(1,2),(3,4),(5,6)]` must return `([1,3,5],[2,4,6])`.

(6) Redefine `splitPair`, called `splitPair'`, using `let` binding.

(7) Define function `temp :: Double -> String` that receives the temperature as input and returns the state of the weather, as follows:

- If temperature is less than 32 degrees, then it says it is freezing.
- If temperature is less than 50 degrees and above 32 degrees, then it says it is cold.
- If temperature is less than 75 degrees and above 50 degrees, then it says it is mild.
- If temperature is above 75 degrees, then it says it is warm.

Use guards to define this function.

Example: `temp 35` may return `"Cold"`.

(8) Define function `abbrevDecoder :: String -> String` that takes an abbreviation as input. If the input string matches a known text message abbreviation, it outputs the unabbreviated form, else outputs: "unknown". The following abbreviations should be supported:

- LOL – laughing out loud,
- IDK – I don't know
- BFF – best friends forever
- IMHO – in my humble opinion
- TMI – too much information

Use guards to define this function.

Example: `abbrevDecoder "BFF"` may return `""best friends forever""`.

(9) Redefine function `abbrevDecoder`, called `abbrevDecoder'`, using case expressions.

(10) Define function `subString :: Int -> Int -> String -> String` that receives the lower index, upper index, and a string as input. It returns the substring starting from the lower index to the upper index (both inclusive). Here are the regulations:

- Lower and upper indexes cannot exceed or be equal to the size of input string. If so empty string must be returned.
- Lower cannot be larger than the upper index. If so empty string must be returned.
- Neither lower index nor upper index can be negative. If so empty string must be returned.

Use guards in your definition.

Note: You cannot use library function from `Data.Strings` and `Data.Text` modules.

Here is an example of running the function on different inputs:

```
ghci> subString 7 19 "hello"
""
ghci> subString 1 19 "hello"
""
ghci> subString 4 2 "hello"
""
ghci> subString (-1) 3 "hello"
""
ghci> subString (-4) (-1) "hello"
""
ghci> subString 2 (-1) "hello"
""
ghci> subString 2 4 "hello"
"llo"
ghci> subString 0 3 "hello"
"hell"
ghci> subString 0 4 "hello"
"hello"
ghci> subString 0 5 "hello"
""
```

(11) Define function `isSubstring :: String -> String -> Bool` that receives two strings as input and returns true if the first string is a substring of the second string. Otherwise, it returns false.

Note: You cannot use any functions from `Data.Strings` and `Data.Text` libraries.

Hints: 1) Empty string is a substring of any string. 2) A non-empty string is not a substring of empty string. 3) You can use `findInd` function from previous lab on functional programming. 4) You can use `subString` from previous question.

Here is an example of running the function on different inputs:

```
ghci> isSubstring "imx" "jimsx"
False
ghci> isSubstring "imx" "jiyimsx"
False
ghci> isSubstring "imx" "jiyimxs"
True
```