Weekly Reading. No weekly reading this week.

Programming Homework: More Haskell Functions

For this assignment you will be implementing Haskell functions in two files called hw9a.hs and hw9b.hs. Each of your function definitions must be accompanied by explicitly declared function types. For each function you must provide two different implementations: one that does not use pattern matching and guards, and one that does use pattern matching and guards. Place the definitions without patterns and guards into hw9a.hs and the definitions with into hw9b.hs. You must write each “from scratch” using only what we have discussed in class. Thus, each function should use recursion and should not call similarly purposed functions that are already provided in Haskell. Be sure to include comments as needed throughout your implementation files, including a file header with your name, the homework number, and so on. For each problem, you must provide multiple test cases in addition to the examples shown below. Your tests must be defined to ensure that your functions work correctly.

1. Write a function myMinimum that returns the smallest of a given list of values. Example: myMinimum [7,1,9,12,10] should return 1. Note the function should return an exception (error) when called on an empty list. Be careful with respect to efficiency, i.e., your implementation must be $O(n)$ for an $n$-element list.

2. Write a function myReverse that takes a list and returns the reverse order of the list. Example: myReverse [1,2,3] should return [3,2,1]. Note that this function does not require guards.

3. Write a function myLength that gives the length of a list. Example: myLength [1,3,5] should return 3. Note that this function does not require guards.

4. Write a function myElement that takes a value and a list and returns true if the value is in the list, and false otherwise. Examples: myElement 3 [1,2,3,4] should return true whereas myElement 3 [1,2,4,5] should return false.

5. Write a function myElements that takes two lists of values and returns true if all the values in the first list are in the second list. Examples: myElements "db" "abcd" should return true whereas myElements [1,2] [0,1,3,4] should return false. Trivially, myElements [] [1,2,3,4,5] is true. Note you can call myElement from within myElements.

6. Write a function myReplace that takes a pair of values and a list and returns a new list such that each occurrence of the first value of the pair in the list is replaced with the second value. Example: myReplace (2,8) [1,2,3,2] should return [1,8,3,8].

7. Write a function myReplaceAll that takes a list of pairs and a list of values and returns a new list where each occurrence of the first value in a pair is replaced by the second value in the pair. The replacement should occur in order of pairs. Examples: myReplaceAll [(’a’, ’b’), (’c’, ’d’)] "abcd" should give "bbdd" and myReplaceAll [(1,2), (2,3)] [1,2,3,4] should give [3,3,3,4]. You can call myReplace from within myReplaceAll. Note also that you do not need guards to define this function.
8. Write a function `myElementSum` that takes a value and a list, and returns the sum of the given values in the list. Examples: `myElementSum 10 [15,10,25]` should return 10, `myElementSum 3 [3,2,3,2,3,4,3]` should give 12 and `myElementSum 3 []` should give 0.

9. Write a function `removeDuplicates` that takes a list of values, and returns the original list with duplicate values removed. Examples: `removeDuplicates ['a','b','a','c','b','a']` should return `['c','b','a']` and `removeDuplicates [10,11,13,11,12]` should return `[10,13,11,12]`. Note you can call `myElement` within your `removeDuplicates` function.

10. Write a `mergeSort` function that takes a list of pairs and sorts the list on the first element of the pair using the merge sort algorithm. For example, `mergeSort [(2,10), (1,15), (4,30)]` should return `[(1,15), (2,10), (4,30)]` and `mergeSort ["b",40), ("c",20), ("a",30), ("d",10)]` should return `[("a",30), ("b",40), ("c",20), ("d",10)]`. Note that you can use the `div` function to perform integer division (e.g., `div 5 2` evaluates to 2—alternatively, you can write `5 `div` 2`).

What to Turn In: You must submit the following on the due date for your assignment to be considered complete. Your submission must be done via your GitHub Classroom repository for HW9. Note that you do not need to include a cover sheet (but can in your write up if you wish), but must follow the guidelines outlined in the cover sheet.

- A single PDF file containing a print out of your answers to questions 1–10, the test cases you developed for each function, example runs showing your functions work for each test case, and a print out of your “discussion”. Be sure to include your full name on the front page of your write up.

- Your `hw9a.hs` and `hw9b.hs` source files.