CPSC 326: Homework #4
Due Tuesday, February 9

Create a file called `hw4.hs` that contains definitions for the following functions. You must implement each of these functions using recursion, but without pattern matching and also “from scratch” (i.e., only using basic Haskell operations and not functions already defined in Haskell that do the same thing). You may, however, use calls to functions you write in this homework to help implement your functions. Each function can (and must) be implemented with what we have learned so far in class, including handling error cases. (Note you should run the similar functions in Haskell to see how error cases should be handled.) For each function, also give an appropriate type definition as part of your code. Hand in a printout of your source code in class on the due date.

1. Write a function `myMin` that returns the smallest of a given list of values. Example: `myMin [7,1,9,10,12]` should return `1`. Be careful with respect to efficiency.

2. Write a function `myRev` that takes a list and returns the reverse order of the list. Example: `myRev [1,2,3]` should return `[3,2,1]`.

3. Write a function `myLen` that gives the length of a list. Example: `myLen [1,3,5]` should return `3`.

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

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

6. Write a function `myRemElem` that takes a value and a list and removes from the list all occurrences of the value. Example: `myRemElem 1 [2,1,3,2,1,4]` should return `[2,3,2,4]`.

7. Write a function `myRepl` 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: `myRepl (2,8) [1,2,3,2]` should return `[1,8,3,8]`.

8. Write a function `mySub` 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: `mySub [(‘a’,‘b’), (‘c’,‘d’)] "abcd"` should give "bbdd" and `mySub [(1,2), (2,3)] [1,2,3,4]` should give `[3,3,3,4]`. Note that you can call `myRepl` from within `mySub`.

9. Write a function `myElemSum` that takes a value and a list, and returns the sum of the occurrences of the given value in the list. Examples: `myElemSum 3 [1,2,3,2,3,4,3]` should give `9` and `myElemSum 3 []` should give `0`.

10. Write a function `myAtIndex` that takes an integer i and a list, and returns the i-th element of the list (assuming indexes are from 0 to length - 1). Example: `myAtIndex 0 [5,1,4,3,2]` should return `5` and `myAtIndex 4 [5,1,4,3,2]` should return `2`.