- Problems are **due by Wed Feb. 21st before class**.
- Problems marked with "**P**" should be implemented in Haskell and a **printout** (comment your functions and give sample outputs[1]) should be turned as part of the assignment.
- If you turn in more than one page, **staple** all of those pages together!!!

(Small) Group Assignment 2

Problem 1 (P, ADT Queue)
Define an abstract data type *Queue* with the following operations:

- *newQ*: returns a new empty queue
- *emptyQ*: given a queue q, returns `True` if q is empty and `False` otherwise
- *enQ*: adds an element to the *end* of the queue and returns the new queue
- *deQ*: removes an element from the *front* of the queue and returns the new queue
- *headQ*: returns the first element of the queue

a) *Specification*: (i) give the *signatures* of these operations in Haskell (make *Queue* a polymorphic type), and (ii) give the *axioms* (equations) that your *Queue* ADT should observe

b) *Implementation*: in Haskell, (i) define the concrete (and polymorphic) data type for *Queue*, and (ii) define the abovementioned functions.

c) As a test output, print the value for `foldr enQ newQ [1..10]`. (Do you see what this does??)

Problem 2 (P, List Reversal using Stack)

a) Define the polymorphic ADT *Stack* in Haskell: the specification can be as given in class, but the implementation should be based on Haskell lists.

b) Use your Stack ADT to implement a function `reverse :: [a] -> [a]` that reverses a list by pushing all elements of the given list on a stack and creates the reversed list by popping the elements from the stack.

Problem 3 (Structural Induction)
The *height* of a tree is defined as the length of the longest path from the root to any leaf.

a) Define a Haskell function `myheight` that works on a data type `BinTree` for binary trees and that returns the height of the tree.

b) Prove the correctness of your implementation `myheight`, i.e., use structural induction to show that for any $bt \in$ `BinTree` we have: $height\,(bt) =$ `myheight` bt.

[1] don't try to fake those outputs if you don't have the solution; rather explain what you have and where the problem is!