PART I. Create a file `hw10.pl` that implements the following API for a binary search tree. Turn in a hardcopy print-out of `hw10.pl` in class by the due date with examples showing your implementation works. A binary tree should be represented using a `node(v, l, r)` structure where `v` is the value of the node, `l` is the left subtree, and `r` is the right subtree. The empty tree should be denoted as `nil`. As in the previous homework, you should try different “binding patterns” for each of your relations (i.e., different arguments as inputs versus outputs).

1. Define a relation `min(X,Y,Z)` such that `Z` is the smaller value of `X` and `Y`. E.g.: `min(2,3,Z)` should return `Z = 2`.

2. Define a relation `insert(X,T1,T2)` that creates a new tree `T2` identical to `T1` but with the new value `X`. E.g.: `insert(4,nil,T)` should return `T = node(4,nil,nil)` and `insert(2,node(4,nil,nil),T)` should return `T = node(4,node(2,nil,nil),nil)`.

3. Define a relation `find(X,T)` that returns true if the value `X` is in the tree `T`. E.g.: `find(2,node(4,node(2,nil,nil),nil))` should return true whereas `find(1,node(4,nil,nil))` and `find(1,nil)` should both return false.

4. Define a relation `height(T,N)` that returns the height of the tree `T`. Note that you should define a `max` relation to use in the implementation of `height` (which can be based on your `min` relation). E.g.: `height(nil,N)` should return `N = 0` and `height(node(4,node(2,nil,nil),nil),N)` should return `N = 2`.

5. Define a relation `delete_it(X,T1,T2)` that creates a new tree `T2` identical to `T1` but without the value `X`. E.g.: `delete_it(2,nil,T)` should return `T = nil` and `delete_it(4,node(4,node(2,nil,nil),nil),T)` should return `T = node(2,nil,nil)`.

PART II. Come up with 5 non-trivial examples of programs using our “little” language. Across the 5 examples you should use each construct at least once. For each program describe what it does. Turn in each of your programs, typed, with the corresponding descriptions of each.