Here is the problem statement for Assignment 5. The due date is **12th March, 2024**. Please submit on Gradescope.

Consider the representation of trees ("pre-terms") using the following data type definition

type *tree* = V of *string* | C **of** { node: *symbol* ; children: *tree* **list** };;

with suitable type representations for types *symbol* and *signature.*

1. Given a signature consisting of symbols and their arities (>= 0) in any suitable form -- either as a list of (symbol, arity) pairs, or as a function from symbols to arities (you choose which) -- write a function *check_sig* that checks whether the signature is a valid signature (no repeated symbols, arities are non-negative etc.)
2. Given a valid signature (checked using *check_sig*), define a function *wftree* that checks that a given tree (pre-term) is well-formed according to the signature.
3. Define functions *ht*, *size* and *vars* that given a well-formed tree, return its height, its size and the set of variables appearing in it respectively. Use *map*, *fold_left* and other such functions as far as possible wherever you use lists. (These have all been defined in class, but come up with representative tests, which can work for the subsequent programs).
4. Write a function *mirror,* which given a tree *t* returns a tree that is the mirror image of *t*. That is, at each level for each node, its children are reversed.
5. Define a suitable representation for substitutions as a table defined as a list of pairs (check that the table is a valid representation of a function). Come up with an efficient representation of *composition of substitutions*.
6. Define the function *subst* that given a tree *t* and a substitution *s,* applies (the Unique Homomorphic Extension of) *s* to *t.* Ensure that *subst* is efficiently implemented. (One version of *subst* has already been defined in class.)
7. Define the function *mgu* that given two well formed trees (terms) *t1* and *t2*, returns their most general unifier, if it exists and otherwise raises an exception *NOT_UNIFIABLE*.
8. Provide at least 8 test cases to show that *mgu* works correctly, clearly showing that your examples cover all cases in the analysis.
9. Show that *mgu (t, u) = mgu (mirror u, mirror t).*