Part 1:

We defined datatypes with the additional parameters mentioned according to the grammar, Then we proceeded to define the translator it was simple for proc-exp and letrec-exp as we only had to return the proc-nested-exp and letrec-nested-exp with the parameters from the letrec-exp and proc-exp but for call exp we had to deal with the count so we would return a diff exp with count symbol as a var-exp and const-exp with value -1 as its two expressions if the rator was a var-exp and returns const-exp with value 1 otherwise.

Then the proc-nested-exp and letrec-nested-exp in interp.scm were also simple but when evaluating the call-nested-exp we had to extract the procedure from the environment to create a new one with a new saved-env that has the value of the new count then we apply the procedure.

In the apply-procedure, we did the same thing we do with normal proc but we evaluate a nested-procedure instead. We also printed the recursive counter before that so that we can return the value of the body.

In environments.scm we initialized count as 0 and when evaluating extend-env we check if we're dealing with a nested proc using cases and we return a proc-val of a new nested-procredure we return val otherwise as it used to do in the default case, also extend-env-rec-nested was the same as extend-env-rec but we return a nested-procedure instead of normal procedure

Part 2:
In the translation var-exp simply returns the name of the var joint with the number that we extract from apply-senv-number
In proc-exp and let exp since we shadow the variables we first check if the number of the variable is greater than 1 then we return the identifier with its number only otherwise we return it with the message of the shadowing

Workload:
We all worked on the project together in a couple of sittings. We solved the questions together on one computer. Ahmed is the one that wrote the code.