COMP301-PROJECT 3
A brief report

Kerem Aksoy (64243)
Ahmet Hakan Hafif(64092)

PART 1

Since each procedure call is evaluated under the case call-exp, we need to handle the requirements of this part inside interp.scm. We simply create(define) a global variable called counter inside interp.scm, and each time a procedure call occurs, we increment it and display its value.

```
(define counter 0) ;This is the global variable which counts the number of procedure calls
```

```
(call-exp (rator rand)
  (let ((proc (expval->proc (value-of rator env)))
        (arg (value-of rand env)))
    ;;PART 1:
    (begin
      (set! counter (+ counter 1)) ;Incrementing the counter since we will call a procedure
      (display (string-append "# of procedure calls: " (number->string counter))) ;Displaying the number of calls
      (newline) ;Going to new line in each display
      (apply-procedure proc arg) ;Applying the procedure
    ) ;End of the begin block
    ))
```

PART 2

The most difficult part was Part 2 of this project. We spent a lot of time in this part.
Writing the translator part is not that much hard since we simply use new expressions such as proc-nested-exp, letrec-nested-exp,call-nested-exp while translating the previous forms into their new forms. This can be handled easily by following the grammar and understanding how they are represented in data structures.(inside translator.scm part)
Also, in the environment part(environments.scm), the key was implementing the extend-env part since it needs to be used changing the name variable of the nested-proc with the var value which is pushed by let expression to the environment.In addition, other important parts in this project were call-exp and apply-procedure part in the interp.scm.
Since we also need to give current value of the count variable to the apply procedure while calculating the call-exp part other than the older version of call-exp, we needed to cons it with the argument of the operator to use it inside the apply procedure part.(interp.scm)
Issue of incrementing the counter is handled in the apply-procedure part and after incrementing it we extend the environment with the new value to update its value. And we give this extended environment the value of body parts as the environment. The reason is that it will use the updated version in case that there are more calls to the same function.
A note: Initially the value of count is zero in the environment, and after each call to the procedure it is incremented by 1.

PART 3

In this part, we first wrote code for a helper function called apply-senv-number which returns the number of occurrences of a variable in the environment.

To meet the requirements of this part, we need to change places where a new variable is declared.Also, we need to change the var-exp part to translate the variables to their appropriate forms whenever a variable(reference) appears. In this language, there are two places such as proc-exp and let-exp where a variable is declared. In addition to that, there is only one place where we handle the reference case which is the var-exp part.
In the declaration part, we need to translate the variables according to their number of occurrences in the environment which is found by apply-senv-number function. For example, when a variable(x) appears for the first time, we need to translate it to x1 form, and all the references inside the scope of this declaration to x need to be x1. So, whenever a variable is declared, we count the occurrences of it and add +1 to have the number part. After this is done, we need to extend the environment while translating the body(scope) of these declarations, we will use apply-senv-number again to have the number part while translating the variables. Since var-exp will be translated in an environment where a new declaration of it is added by the expression which declares the variable, it is enough to use apply-senv-number while calculating the number part in the var-exp part.

Workload
While doing that project, we sit side by side and try to implement this project. So, it can be said that it was a teamwork and the workload was equally distributed among us.

All of the project has passed all the test cases.

A note:
There are also some comments in the code to show the way how it is implemented.