Part 1:

1. “let” in L3 is a special form, because it consists of a special operator on the left, and the calculation of the expression is performed uniquely.
2. Because that applications are computed by substituting computed values into the body of the closure, we need the function valueToLitExp to make the types fit – computed values of parms must be turned back in Literal Expressions that eval to the computed value.
3. The function is not needed because when applying normal evaluation, we only evaluate the operator hence all the operands remain as CExpressions which than substituted as is.
4. The use of the data structure of the environment frees us from the need to substitute the arguments within the body of the procedures, so we don’t need the valueToLitExp function.
5. Normal evaluation is better because:
   - There may be cases which normal is more efficient (evaluation of a compound if-exp when there is no need to calculate the alt because the test is always "true".
   - There may be cases which we could avoid runtime errors.
   - There may be cases which we could avoid infinite loops.

   (L3 -

   (define loop (lambda (x) (loop x)))

   (define g (lambda (x) 5)) -

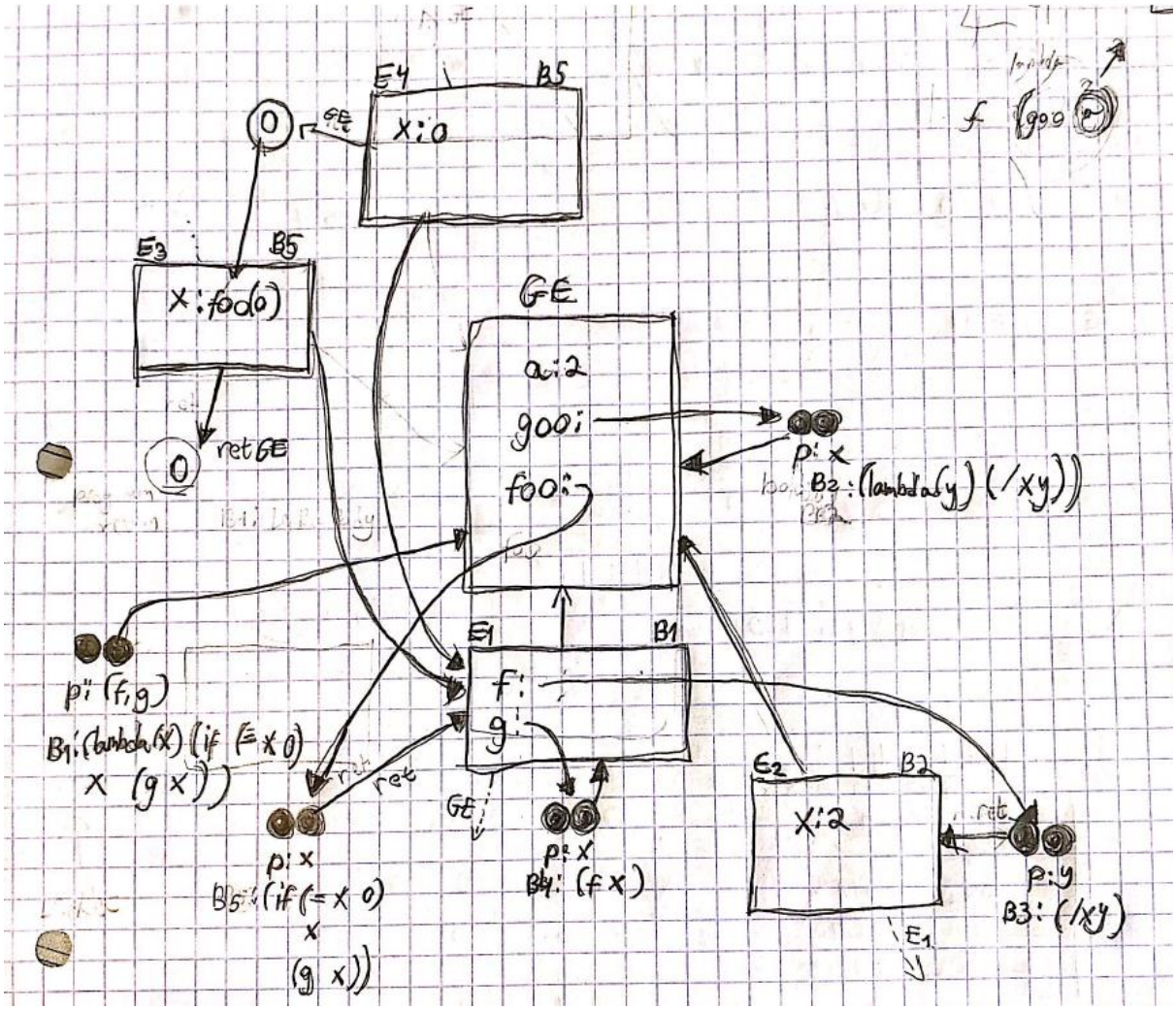
   (g (loop 0))

6. The reason that we use "applicative" instead of "normal' is because of efficiency, for instance in case we have app-exp which gets a calculation as a parameter and this parameter occurs several times in the body of the function, we would need to calculate this exp several times (twice in normal) instead of calculating once (applicative).

   ((
   (lambda (x) (+ x x)
   (* 2 3))
)

7. A. The naming is merely cosmetic as the guarantee of a lack of free variables ensures that we will not have a case in which a given variable will match another variable after substitution, as by definition, no variables will remain after substitution, therefore no collision can occur.
   B. we will change only the function apply-closure because only there we perform renaming.

```
const applyClosure = (proc: Closure, args: Value[], env: Env): Result<Value> =>{
    const vars = map((v: VarDecl) => v.var, proc.params);
    const litArgs = map(valueToLitExp, args);
    return evalSequence(substitute(proc.body, vars, litArgs), env);
```

8. 
E4
B5
x:0
E3
B5
x:foo(0)
GE
a:2
goo:
foo:
E1
B1
f:
g:
E2
B2
x:2
p:x
B2: (lambda(y) (/ x y))
p:(f,g)
B1: (lambda(x) (if (= x 0)
x (g x)))
p:x
B5: (if (= x 0)
x
(g x))
p:x
B4: (f x)
p:y
B3: (/ x y)
ret
GE
E1