Part 1: Theoretical Questions

1. the function value2LitExp converts a variable “val” and turns it into a literal expression with that value.
   when applying the applyProc function, which receives arguments of type value, the body is a list of cExp expressions. we need to replace all varRefs in the body with the corresponding values. This might create a problem, for example when we try to replace a value that is a literal expression (let's say the number 3). the resulting body is not a valid AST. therefore we must map the values to a corresponding expression. This mapping is performed by the function val2LitExp.

2. Because in the normal evaluation strategy arguments are not evaluated before they are passed to closures. Therefore, the applyClosure receives as arguments CExps and replaces VarRefs in the body with CExps which is correct according to the type definition of CExp (unlike applicative order evaluation). We do not need to turn the values back into expression before we apply the substitution in the body, since the rands are passed as non-evaluated expressions.

3.

- Evaluating the expression directly. First, evaluating the bindings. Then, extending the environment with their values. And finally evaluating the body with the extended environment.

- Converting the Let expression into a procedure application and then evaluating that.

4.

- Invalid variable type pass to procedure:
  Example**:** (define true #t)

- Arithmetic errors
  Example**:** (/ 1 0)

- Wrong number of variables:
  Example: ((lambda (x) (+ x 1)))

- Apply procedure in wrong way:
  Example: (1 + 2) =>

5.

A special Form is an expression. For each special form - a special evaluation rule exists.
Primitive Operator is an operator that is already defined in the interpreter. All Primitive Operators have the same evaluation rule.

6.

In the **substitution model,** each activation of a procedure (even if it has been activated before) involves rewriting its body code. That operation may be very expensive in running time and memory.

In the **environment model**, in each operation of a procedure, instead of rewriting it as in the substitution model, we will define a new frame, in which the bindings of the parameters of the procedure with the values sent for them will be defined. This frame will be added to the hierarchy, meaning it will be linked to one of the existing frames.

For example:

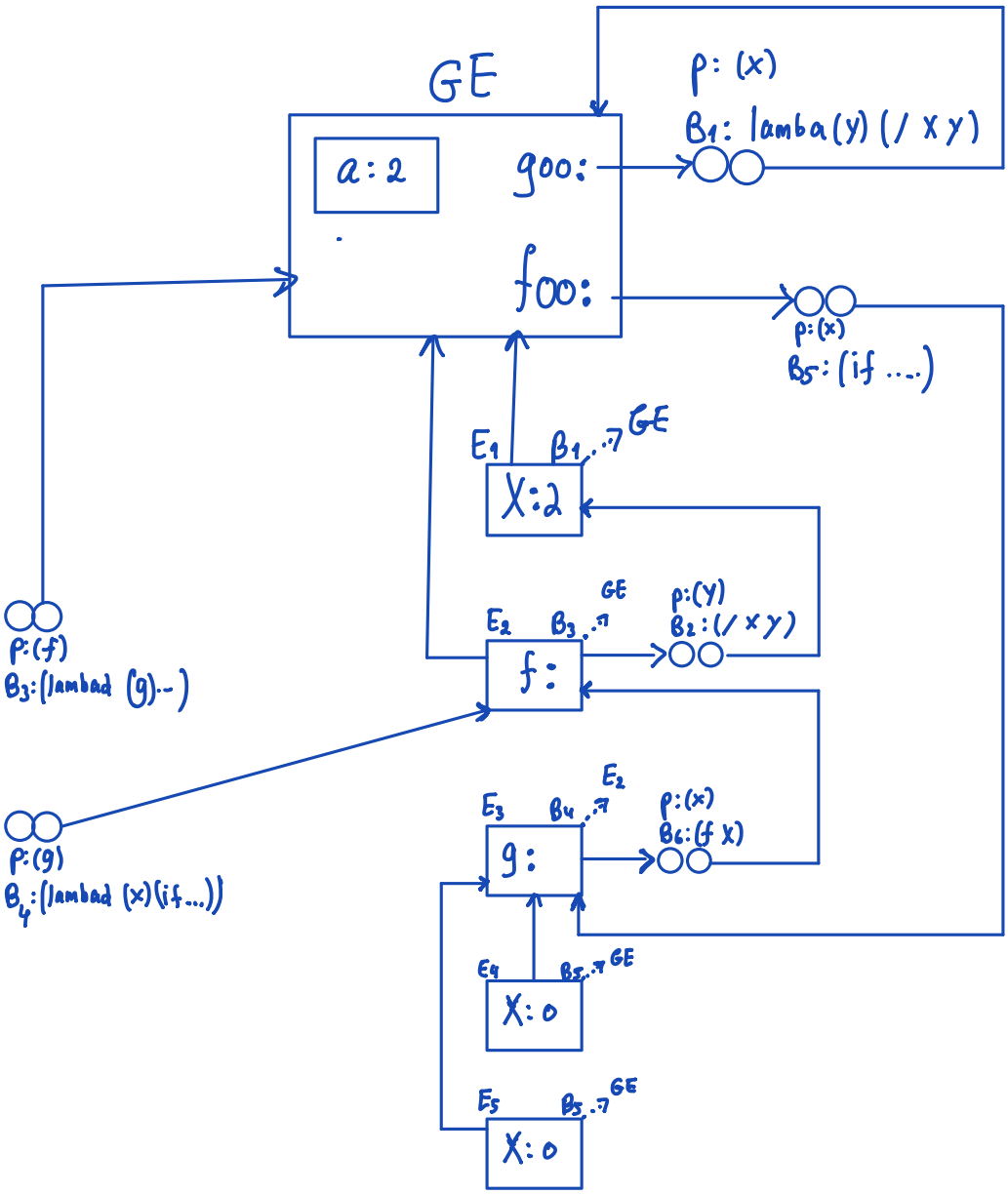
```
(define fact
        (lambda (n)
                (if (= n 1)
                        1
                        (* n (fact (- n 1))))))
        (fact 2)
```

```
=> (fact 2)
=> ((lambda (n) (if (= n 0) 1 (* n (fact (- n 1))))) 2)
…
```

7.
We want to support mutation in the language we implement. Note that at this point, all the variables are immutable - we can create a new variable and bind it to a value, but we cannot change its value. In some cases, we want to be able to change the value of an existing variable.

In the **box environment model**, we use boxing to enable mutation. We saw in the lecture why we needed mutation to properly model recursion (letrec) and model the global environment with forward usage of global variables and global mutual recursive procedures.

1.8)


GE
a: 2
goo:
foo:
ρ: (x)
B1: lamba(y) (/ x y)
ρ:(x)
B5: (if ....)
E1
B1
GE
X:2
E2
B3
GE
f:
ρ:(y)
B2: (/ x y)
ρ:(f)
B3: (lambad (g)...)
E3
B4
E2
g:
ρ:(x)
B6: (f x)
ρ:(g)
B4: (lambad (x)(if...))
E4
B5
GE
X: 0
E5
B5
GE
X: 0

2.1.3

bound? Expression as other special forms execute operations that require language-level constructs, something that can't be achieved by primitive or a user function.

In addition, The main difference between special forms and procedures is that procedures have all their arguments evaluated before it's applied, therefore, bound cannot be a user function.

2.2.2

Time Expression as other special forms execute operations that require language-level constructs, something that can't be achieved by primitive or a user function.

In addition, The main difference between special forms and procedures is that procedures have all their arguments evaluated before it's applied, therefore, bound cannot be a user function.