Q1:

No, the let expression is not a special form.
The reason is that Let expression does not have a specific evaluation rule, while the special forms have their own special rules to evaluate and apply the arguments.

Q2:

No, in the evaluation of a let expression, a new local environment with bindings for the let variables is created.
The pointer for the environment points to the current environment, after that the Evaluateion of the body of the let happens in this new environment.

Q3:

Semantic errors:

1) unknown references: (defineee x 7), the error happens because there is no reference to defineee.

2) Use of a non-initialized variable: (define x y), y is not initialized so it can't be used.

3) type incompatibility: (define x "y") (define k (+ 7 x)), x is expected to be a number but is defined as something else.

4) Errors in expressions: (define x7), the use of a bad syntax results to and error, the compiler doesn't know what we actually want and sees the bad syntax as an error.

Q4:

Purpose:

Transform a value into a literal expression denoting this value, turn back the values of type Value to the expressions.

Problem:

we use it when the applications are computed by substituting values in the body of the closure, and we need to fit the types of the expressions, so we don't create some errors, so we return the values to fit the types.

Q5:

because the arguments are evaluated after the substitution.
That guarantees that there won't be any problem with types when we evaluate them, so its not needed.

Q6:

The main difference between special forms and primitive operators are that primitive operators have all their arguments evaluated before it's applied. That's not true for special

forms, where they have a set of rules to follow in regard of evaluating and applying the arguments. special forms are used to do non primitive evaluations that the compiler is unfamiliar with.

Q7:

1) unlike the substitution model, we don't need to rename the variables into unique names, that's because of the pecking order structure of the frames.

2) When we use the data structure of the environment model, we don't need to place the arguments in the body of the procedures.

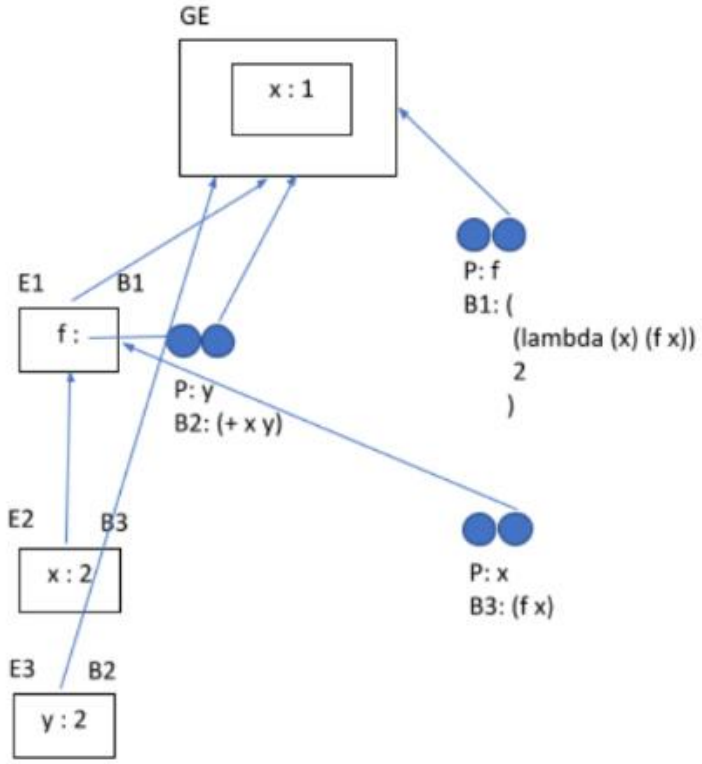
Example:

```
(define x 1)
(
 (lambda (f) ( (lambda (x) (f x)) 2 )))
 (lambda (y) (+ x y))
)
```

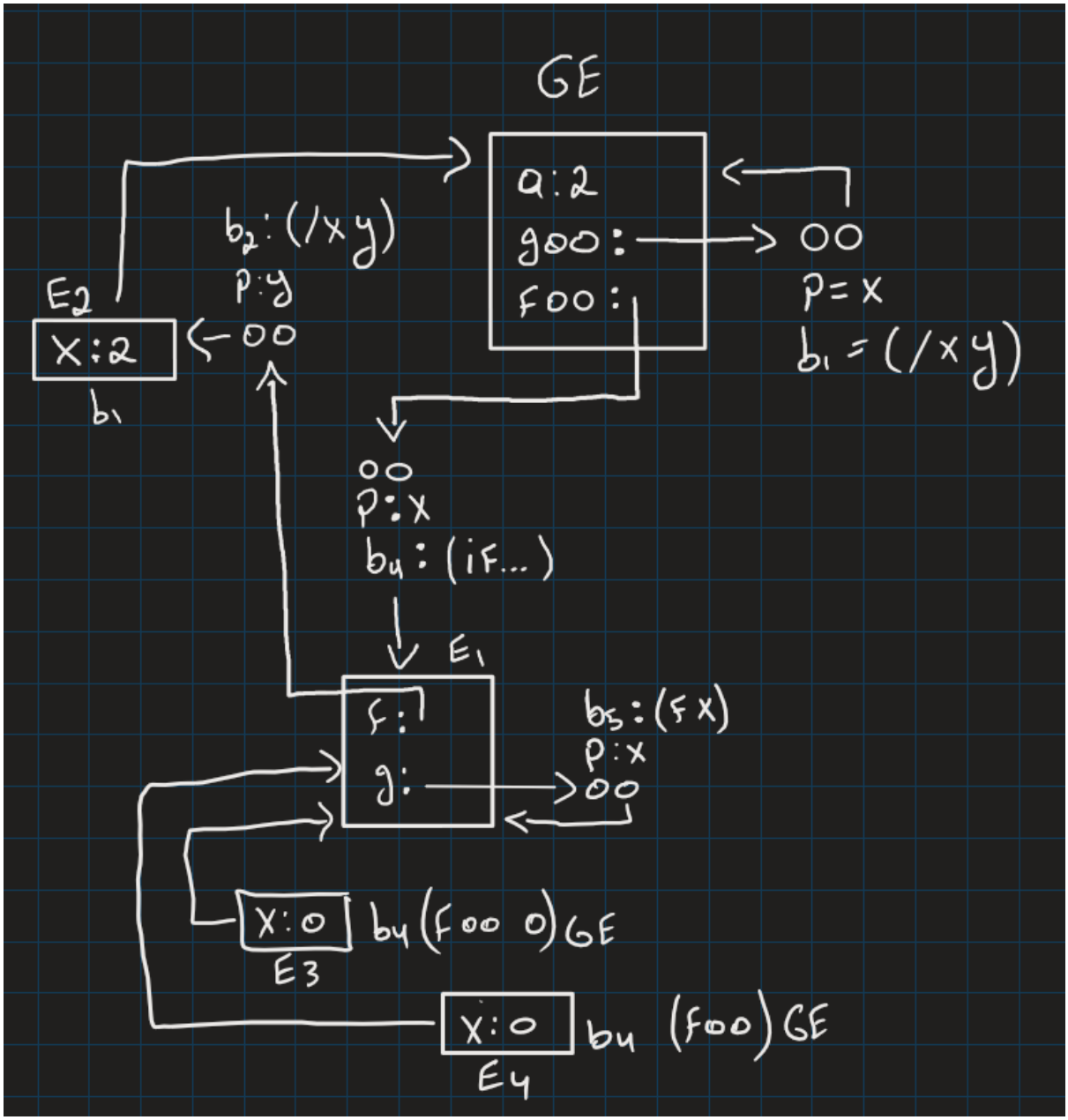
substitution model:

```
(define x 1)
(
 (lambda (f) ((lambda (x1) (f x1)) 2)))
 (lambda (y) (+ x y))
)
```

environment model:



Q8:


GE
a:2
goo:
foo:
OO
p=x
b1 = (/x y)
b2: (/x y)
p:y
E2
x:2
b1
OO
p:x
b4: (if...)
E1
f:
g:
b5: (f x)
p:x
OO
x:0
E3
b4 (foo 0) GE
x:0
E4
b4 (foo) GE