עבודת בית 3

מגישים: עומרי גולדברג 208938985, נעה חרדון 312164668

Part 1:

1. Yes, let is a special form in L3. Let form is a special form because it is not evaluated in a standard form like AppExp, the interpreter of L3 must rewrite let into an application, this means that there is a special rule before the syntactic form of let expressions is evaluated.

2. the role of the function valueToLitExp is to provide a better type definition for ASTs, it substitute values to their expression like, number -> NumExp and etc. it helps us to compute the closures.

3. The valueToLitExp function is not needed in the normal evaluation strategy interpreter because arguments are not evaluated before they are passed to closures.
   in normal evaluation strategy the substitution happens before the evaluation of the arguments, ValueToLitExp is needed when there is need to calculate closures using variables that are substituted with values.

4. the valueToLitExp function is not needed in the environment model interpreter because there is no substitution in that model and the valueToLitExp only needed when we need to substitute variables and values (as mentioned in question 3). environment model is using VarRef and new environments to compute.

5. we will switch from applicative order to normal order evaluation when we do not need the ValueToLitExp, or to get faster execution by avoiding unnecessary calculations, or when using applicative evaluation can throw en error. For example:

```
(L3
(define loop (lambda (x) (loop x)))
(define g (lambda (x) 5))
(g (loop 0)))
```

   In this example, in normal order the application (loop 0) is not evaluated and in applicative order the call (g (loop 0)) enters an infinite loop.

6. we will switch from normal order to applicative order evaluation when we want to avoid evaluation of expressions more than once (in these cases applicative will be faster). For example:

```
(define square (lambda (x) (* x x)))
(define sum-of-squares (lambda (x y)
                         (+ (square x) (square y))))
(define f (lambda (a) (sum-of-squares (+ a 1) (* a 2))) (f 5)
```

   In normal evaluation, we will evaluate the expression "(+ 5 1)" in the expression "(* (+ 5 1) (+ 5 1))" twice, whereas in applicative evaluation order we will evaluate "(+ 5 1)" directly into the number 6.

7.

a. let c be a closed closure and let $x_1, x_2, \dots, x_n$ be the arguments in this closed closure c. now, assume by contradiction that renaming is required. it means that there is an $1 \leq i \leq n$ such that x_i has two different definitions, and that this x_i is present in the body of the closure (elsewhere the renaming was not required). then x_i is a free variable, in contradiction to the initial assumption.

b. the evaluation rules for naïve substitution should be as following:
   - identification of the syntactic structure of the expression in the top level
   - identification of the immediate sub expressions of the expression
   - evaluate using the right rule for the construct of the expression (in these rules we will show it on closure)
     - evaluate using the right evaluation (primOp or closure)
     - in a closure situation, should be computation of the parameters and checking for free variables in the body of the closure.
     - if there are no free variables, make procExpression with the current naming.
     - otherwise, should use renaming and then make procExpression.
     - In a recursive way compute the rest of the expressions in the body.

8.