Assignment 3 – Part 1

1. let expression in L3 a special form? elaborate.

let expression in L3 is **not** a special form. The interpreter in L3 interprets let expression into an application, so it is evaluated like an application expression. It doesn't support evaluation of let expression directly but rewrites it to an application.

2. Is a closure created during the evaluation of a let expression? Refer to the various strategies discussed in class and in the practical session.

Yes, when the parser gets LetExp it transform it from LetExp to AppExp (as we seen in class and practical sessions – transform let to lambda form), when it's operator it from type ProcExp. When the transform Let is evaluate, closure created as part of the evaluation of the ProcExp.

3. List four types of semantic errors that can be raised when executing an L3 program - with an example for each type.

1. Invalid variable type pass to procedure.
   Example:
   (define square (lambda (x) (* x x)))
   (define true #t)
   (square true) => all symbols are valid (parsing pass) but semantic error occurs, square expects number but gets boolean instead.

2. Incorrect calculations.
   Example:
   (/ 1 0) = > all symbols are valid (parsing pass) but semantic error occurs, dividing by zero is not legal.

3. None exist variables.
   Example:
   (define ID (lambda (x) x))
   (ID var) => all symbols are valid (parsing pass) but semantic error occurs, there is no reference to variable "var" unbound identifier).

4. Apply procedure in wrong way:
   Example:
   (1 + 2) => all symbols are valid (parsing pass) but semantic error occurs, the interpreter expects prefix notation so it tries to interpret "1" as applying procedure which is not correct.

4. What is the purpose of valueToLitExp? What problem does it solve?

The purpose of valueToLitExp is to get a variable 'val' (which is not void) and makes the value of it into a literal expression denoting this value.

The applyProc procedure receives arguments which are all of type Value. The body of the closure is a list of CExp expressions. Our objective is to replace all VarRef occurrences in the body with the corresponding values of the arguments. There is a typing problem with this operation - we might want to replace a value such as 5 with a VarRef x which is an expression. If we replace (VarRef x) with the value 5 (a number), the resulting body is not a valid AST. To address this discrepancy, we must map the values of the arguments to corresponding expressions. This mapping is performed in our interpreter with the valueToLitExp function.

5. **valueToLitExp is not needed in the normal order evaluation strategy interpreter (L3-normal.ts). Why?**

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.

6. **What is the difference between a special form and a primitive operator?**

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

7. **What is the reason for switching from the substitution model to the environment model? Give an example**

In the substitution model, each activation of procedure (even if it has been activated before) involves rewriting its body code:
• Calculation of argument values (immediately in Applicative, or if necessary, in Normal)
Rename all the parameters of all the procedures defined in the body
• Applicative - Returns the values of the arguments to the structure of expressions.
• Placing the arguments in each appropriate VarRef.
Heavy operations that require 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.
Using environments releases us from the need to place the arguments inside the body of the procedures.
In addition, the hierarchical structure of the environments releases us from the need to change the names of the variables to unique.

8. Draw an environment diagram for the following computation. Make sure to include the lexical block markers, the control links and the returned values.