**Theoretical Part**

1. Is let in L3 a special form? Justify your answer.

Yes. According to special form’s definition, an expression is a “special form” if it is evaluated in a non-standard way (not as the application rule).

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.

1. In L3, what is the role of the function valueToLitExp?

valueToLitExp provides a better and uniform type definition for the ASTs.

It wraps a specific value with its corresponding type expression.

This function is needed when we want to substitute variables with values when a closure is computed.

1. The valueToLitExp function is not needed in the normal evaluation strategy interpreter (L3-normal.ts). Why?

As written above, valueToLitExp is needed when there is a need to substitute variables with values.   
In normal evaluation we make the substitutions before evaluating the arguments, thus, substituting variables with expressions rather than values. Hence, valueToLitExp is not needed.

1. The valueToLitExp function is not needed in the environment-model interpreter. Why?

In the environment model there isn’t any substitution at all. As written above, valueToLitExp is needed when there is a need to substitute variables with values, therefore, it is not needed.

1. What are the reasons that would justify switching from applicative order to normal order evaluation? Give an example.

One would change from applicative order to normal order in case applicative order crashes. For instance:

1. (define div0 (lambda (x) (/ 3 0)))
2. (define foo (lambda (x y) (if (= x 0) x y)))
4. >(foo 0 (div0))
6. Normal order:
8. normal-eval: [(foo 0 (div0))]
9. normal-eval: [((lambda (x) (if (= x 0) x y)) 0 (div0))]
10. normal-eval: [(lambda (x) (if (= x 0) x y))] ==>
11. <closure (x) (if (x) x y)>
13. substitute:
14. (if (x) x y) o {x = 0} ==> (if (= 0 0) 0 y)
15. (if (= 0 0) 0 y) o {y = (div0)} ==> (if (= 0 0) 0 (div0))
17. reduce:
18. normal-eval: [(if (= 0 0) 0 (div0))]
19. normal-eval: [(= 0 0)]
20. normal-eval: [ = ] ==> <primitive proc =>
21. normal-eval: [ 0 ] ==> 0
22. normal-eval: [ 0 ] ==> 0
23. ==> #t
24. normal-eval: [ 0 ] ==> 0
25. ==> 0
27. Applicative order:
29. applicative-eval: [(foo 0 (div0))]
30. applicative-eval: [((lambda (x) (if (= x 0) x y)) 0 (div0))]
31. applicative-eval: [(lambda (x) (if (= x 0) x y))] ==>
32. <closure (x) (if (x) x y)>
34. applicative-eval: [ 0 ] ==> 0
35. applicative-eval: [(div0)]
36. applicative-eval: [(lambda (x) (/ 3 0))] ==>
37. <closure (x) (/ 3 0)>

40. applicative-eval: [(/ 3 0)]
41. applicative-eval: [ / ] ==> <primitive proc =>
42. applicative-eval: [ 3 ] ==> 3
43. applicative-eval: [ 0 ] ==> 0
44. ==> Division by 0 ERROR

1. What are the reasons that would justify switching from normal order to applicative order evaluation? Give an example.

One would switch from normal order to applicative order evaluation to improve efficiency. For instance:

1. (define ten (+ 5 5))
2. (define multi5 (lambda (x) (+ x x x x x)))
3. > (multi5 ten)
5. Normal order:
7. normal-eval [(multi5 ten)]
8. normal-eval [((lambda (x) (+ x x x x x)) (+ 5 5))]
9. normal-eval [(lambda (x) (+ x x x x x))] ==> <closure (x) (+ x x x x x)>
10. substiute:
11. (+ x x x x x) o {x = (+ 5 5)}
12. ==>
13. (+ (+ 5 5) (+ 5 5) (+ 5 5) (+ 5 5) (+ 5 5))
15. reduce:
16. normal-eval:
17. [(+ (+ 5 5) (+ 5 5) (+ 5 5) (+ 5 5) (+ 5 5))]
18. normal-eval: [ + ] ==> <primitive proc +>
19. normal-eval: [(+ 5 5)]
20. normal-eval: [ + ] ==> <primitive proc +>
21. normal-eval: [ 5 ] ==> 5
22. normal-eval: [ 5 ] ==> 5
23. ==> 10
24. ...... evaluting (+ 5 5) 4 more times
25. ==> 50
27. Applicative order:
29. applicative-eval [(multi5 ten)]
30. applicative-eval [((lambda (x) (+ x x x x x)) (+ 5 5))]
31. applicative-eval
32. applicative-eval [(+ 5 5)]
33. applicative-eval [ + ] ==> <primitive proc +>
34. applicative-eval [ 5 ] ==> 5
35. applicative-eval [ 5 ] ==> 5
36. ==> 10
38. substitute:
39. (+ x x x x x) o {x = 10} ==> (+ 10 10 10 10 10)
41. reduce:
42. applicative-eval [(+ 10 10 10 10 10)]
43. applicative-eval [ + ] ==> <primitive proc +>
44. applicative-eval [ 10 ] ==> 10
45. applicative-eval [ 10 ] ==> 10
46. applicative-eval [ 10 ] ==> 10
47. applicative-eval [ 10 ] ==> 10
48. applicative-eval [ 10 ] ==> 10
50. ==> 50
52. What are the reasons that would justify switching from normal order to applicative order evaluation? Give an example.