**Principles of Programming Languages**

Assignments 2

1. special form (define, if, etc) are not evaluated when primitive operators are evaluated to their pre-defined denoted value.   
   example: (lambda (x) (+ x x)) the var x is not evaluated.
2. (+ (+ 1 2) (+ 3 4)), (> 1 (+ 2 3)).
3. No, instead of using variables like (define pi 3.14) in L0 we’ll have to replace each use of pi to 3.14.
4. No, instead of using variables like   
   (define square (lambda (x) (\* x x))) in L20 we’ll have to replace each use of square to (lambda (x) (\* x x)).
5. L3:
   1. Map – parallel
   2. Reduce – sequential
   3. Filter – parallel
   4. All – the pred check can be parallel but the last check if all of the item in the list returned true is sequential.
   5. Compose - sequential
6. Value – 3 + 4 + 5 = 12, b is bound to class variable not to (define b 1), so in this case b = 4, the call to f is with new variable c that is bound to lambda and not to (define c 2), so c = 5.

Question 2:

Q. 2.1 –

; Signature: append(lst1, lst2)

; Type:[List \* List -> List]

; Purpose: gets two lists and returns their concatenation

; Pre-conditions: lst1 is a list, lst2 is a list

; Tests: (append ‘(1 2) ‘(#t 4)) → ‘(1 2 #t 4)

Q. 2.2 –

; Signature: reverse(lst)

; Type:[List -> List]

; Purpose: gets a list and reverses it

; Pre-conditions: lst is a list

; Tests: (reverse ‘(1 2 #t 4)) → ‘(4 #t 2 1)

Q. 2.3 –

; Signature: duplicate-items(lst1, lst2)

; Type:[List \* List(Number) -> List]

; Purpose: gets two lists – *lst1* , *lst2* - and duplicates each item of *lst1* according to the number defined in the same position in *lst2*

; Pre-conditions: lst1 is a list, lst2 is a list of numbers

; Tests: ((duplicate-items '(1 2 3) '(2 1 0 10 2))→ '(1 1 2)

Q. 2.4 –

; Signature: payment(sum, lst)

; Type:[Number \* List(Number) -> Number]

; Purpose: gets a sum of money and list of available coins, and returns the number of possible ways to pay the money with these coins

; Pre-conditions: sum >= 0, lst is a list of numbers

; Tests: (payment 5 ‘(1 1 1 2 2 5 10)) → 3

Q. 2.5 –

; Signature: compose-n(f, n)

; Type:[(T->T) \* Number -> T]

; Purpose: gets an unary function *f* and a number *n* and returns the closure of the n-th self-composition of f

; Pre-conditions: f is an unary function ,n > 0

; Tests: (define mul8 (compose-n (lambda (x) (\* 2 x)) 3)) (mul8 3) → 24