Principles of Programming Languages
Function Programming Language (Scheme)

Lab-3 **18/11/2014**

Objective:

- Recursion.

Complete the following recursive programs. See the trace of the output of each program by typing

>> (require racket/trace)

>> (trace function-name)

(1) Insertion Sort

```
(define (insert n list)
   (cond
      [(empty? list) (cons n list)]
      [(<= n (first list)) (cons n list)]
      (else
         (cons (first list) (insert n (rest list))) )))

(define (insertion-sort list)
   (cond
      [(empty? list) list] ;;if the list is empty than the numbers are sorted
      (else
         (insert (first list) (insertion-sort (rest list))) )))
```

(2) Binary search

```
(define binary-search
        (lambda (ls value low high)
                (let ((mid (floor (/ (+ low high) 2))))
                        (cond
                                ((> low high) -1)
                                ((= (list-ref ls mid) value) mid)
                                ((> (list-ref ls mid) value) (BLANK)))
                                ((< (list-ref ls mid) value) (BLANK))))))
```

Tail Recursion

Try the following three tail recursive and non-tail recursive version of the functions. Trace the output to see the difference.

	Tail recursive factorial function	**Non tail recursive factorial function**
1	(define (fact1 n) (if (< n 1) 1 (* n (fact1 (sub1 n)))))	(define (fact2 n) (fact2.1 n 1)) (define (fact2.1 n accumulator) (if (< n 1) accumulator (fact2.1 (- n 1) (* accumulator n))))
2	(define (fib n) (if (< n 2) n (+ (fib (- n 1)) (fib (- n 2)))))	(define (fib2 n) (cond ((= n 0) 0) ((= n 1) 1) (#t (fib-tr n 2 0 1)))) (define (fib-tr target n f2 f1) (if (= n target) (+ f2 f1) (fib-tr target (+ n 1) f1 (+ f1 f2))))
3	**Function to reverse the list** (define rev1 (lambda (lst) (cond ((null? lst) '()) (else (append (rev1 (cdr lst)) (cons (car lst) '())))))))	**Non tail recursive version to reverse the list.** (define (rev2 list) (rev2.1 list empty)) (define (rev2.1 list reversed) (if (null? list) reversed (rev2.1 (cdr list) (cons (car list) reversed))))

Nameless Recursive Functions

Nameless recursive functions are realized using Y combinators.

$$Yt = t(Yt)$$

Example1

Y =(((lambda (fun) ((lambda (F) (F F)) (lambda (F) (fun (lambda (x) ((F F) x))))))

t_{fact} = (lambda (factorial) (lambda (n) (if (= n 0) 1 (* n (factorial (- n 1)))))) 4)

Y t_{fact} is given below.

```
(((lambda (fun) ((lambda (F) (F F)) (lambda (F) (fun (lambda (x) ((F F) x))))))
 (lambda (factorial) (lambda (n) (if (= n 0) 1 (* n (factorial (- n 1)))))) 4)
```

Example 2

t_{fib} = (lambda (fib) (lambda (i) (if (< i 2) i (+(fib(- i 1)) (fib (- i 2)))))) 10)
See the output for **$Yt_{fib.}$**

Example 3

Give **$t_{reverse}$** which will take list as input and reverse the given list

Evaluation Component

A tail recursive version of a function TRlength which compute the length of a list is given, give a non-tail recursive and a nameless version of the function TRlength. **2.5+2.5 =5M**

Tail Recursion
(define (**TRlength** lst) (cond ((null? lst) 0) (#T (+ 1 (**TRlength** (cdr lst))))))

Name the non tail recursive function as **NTRlength** and it's helper function as **NTRlength.1**. The nameless function should take a list as input to output it's length.

Write all your functions in the file named **idnumber.rkt** and upload in Nalanda.