Nalongsone Danddank Student ID : 14958950 StarID: jf3893pd

Email: [nalongsone.danddank@my.metrostate.edu](mailto:nalongsone.danddank@my.metrostate.edu)\

**Metropolitan State University**

**ICS-365-01 —Organization of Programming Languages**

**Homework #7**

1. Find definitions of the Scheme functions EVAL and APPLY, and explain

their actions.

Answer:

- EVAL function takes a “quoted” expression or definition and evaluates it.

And we can use EVAL to force the interpreter to regard expression as an expression in Scheme syntax. EVAL is the opposite of “quoted”.

Example: > (eval ‘(+ 2 3)) => 5 .

- APPLY function evaluates to the result of passing arguments all at once to function. The entails a single call to the function. It is a useful programming trick for define a function that takes a list of arguments, if have available a function that arbitrary number of arguments.

Example: > (apply + ‘(1 2 3 4)) => 10

1. If Scheme were a pure functional language, could it include DISPLAY?

Why or why not?

Answer: It cannot includes DSPLAY function. Because: pure function languages are those that return same result every time that same expression is evaluated. So, this means it cannot do anything would alter the result or create a side effect.

1. What does the following Scheme function do?

(define (x lis)

(cond

((null? lis) 0)

((not (list? (car lis)))

(cond

((eq? (car lis) #f) (x (cdr lis)))

(else (+ 1 (x (cdr lis))))))

(else (+ (x (car lis)) (x (cdr lis))))))

Answer: First define the function name *x* and it argument name *lis.*

Then, inside the function we have condition are:

If the argument *lis* is null then return 0.

If the first element of *lis* is **not** type “list” then go to inside this statement condition:

If the first element of *lis* equal to *#f* (boolean false) then go recursive *x* function by taking the left of lis that begin from second element of *lis* as argument.

Else return the sum of 1 and the result of recursive x function by taking the left of *lis* that as argument.

Else return the sum between recursive x function with the first element of *lis*  and recursive x function with the left of *lis* begin from second element as argument.

The Pseudocode as python programming language like:

def x(lis):

If lis is null :

return 0

If lis[0] is not list :

If lis[0] == FALSE:

return x(lis[1 to end])

Else:

return 1 + x(lis[1 to end])

Else:

return x[lis[0]] + x(lis[1 to end])

1. Write a Scheme function that returns the reverse of its simple list parameter

Solution:

(define (reverse lst)

(reverse-helper lst '()))

(define (reverse-helper lst lst\_help)

(if (null? lst) lst\_help

(reverse-helper (cdr lst) (cons (car lst) lst\_help))))

Screen shot cut Result’s picture:

