**Note: All the scheme expression has been run using the eclipse and Following output was produced.**

(define member?

(lambda (item lst)

(cond

((null? lst) #f)

((equal? item (car lst)) #t)

(else (member? item (cdr lst)))

)))

>

(member? 3 '(1 2 3))

member?( 3, [2, 3])

member?( 3, [3])

**PROGRAM RESULT >>> #t**

>

(member? 'b '(a (b c) d))

member?( b, [[b, c], d])

member?( b, [d])

member?( b, [])

**PROGRAM RESULT >>> #f**

>

(define remove-last

(lambda (item lst)

(cond

((null? lst) '())

((and (equal? item (car lst)) (not (member? item (cdr lst)))) (cdr lst))

(else (cons (car lst) (remove-last item (cdr lst)))))

))

>

(remove-last 'a '(b a n a n a s))

remove-last( a, [a, n, a, n, a, s])

member?( a, [n, a, n, a, s])

member?( a, [a, n, a, s])

remove-last( a, [n, a, n, a, s])

remove-last( a, [a, n, a, s])

member?( a, [n, a, s])

member?( a, [a, s])

remove-last( a, [n, a, s])

remove-last( a, [a, s])

member?( a, [s])

member?( a, [])

**PROGRAM RESULT >>> [b, a, n, a, n, s]**

>

(remove-last '(a b) '(a b (a b) a b (b a) a b (a b) a b))

remove-last( [a, b], [b, [a, b], a, b, [b, a], a, b, [a, b], a, b])

remove-last( [a, b], [[a, b], a, b, [b, a], a, b, [a, b], a, b])

member?( [a, b], [a, b, [b, a], a, b, [a, b], a, b])

member?( [a, b], [b, [b, a], a, b, [a, b], a, b])

member?( [a, b], [[b, a], a, b, [a, b], a, b])

member?( [a, b], [a, b, [a, b], a, b])

member?( [a, b], [b, [a, b], a, b])

member?( [a, b], [[a, b], a, b])

remove-last( [a, b], [a, b, [b, a], a, b, [a, b], a, b])

remove-last( [a, b], [b, [b, a], a, b, [a, b], a, b])

remove-last( [a, b], [[b, a], a, b, [a, b], a, b])

remove-last( [a, b], [a, b, [a, b], a, b])

remove-last( [a, b], [b, [a, b], a, b])

remove-last( [a, b], [[a, b], a, b])

member?( [a, b], [a, b])

member?( [a, b], [b])

member?( [a, b], [])

**PROGRAM RESULT >>> [a, b, [a, b], a, b, [b, a], a, b, a, b]**

>

(define same-structure?

(lambda (x y)

(cond

((and (null? x) (null? y)) #t)

((null? x) #f)

((null? y) #f)

((and (pair? (car x)) (pair? (car y)))

(and (same-structure? (car x) (car y))

(same-structure? (cdr x) (cdr y))))

(else (and (same-type? (car x) (car y))

(same-structure? (cdr x) (cdr y)))))

))

>

(define float?

(lambda (x)

(and (real? x) (not (integer? x)))

))

>

(define same-type?

(lambda (x y)

(or (and (symbol? x) (symbol? y))

(and (integer? x) (integer? y))

(and (float? x) (float? y))

(and (boolean? x) (boolean? y))

(and (char? x) (char? y))

(and (string? x) (string? y)))

))

>

(same-structure? '(1 (a (b 3.14) ((c)))) '(3 (z (x 1.23) ((q)))))

same-type?( 1, 3)

same-structure?( [[a, [b, 3.14], [[c]]]], [[z, [x, 1.23], [[q]]]])

same-structure?( [a, [b, 3.14], [[c]]], [z, [x, 1.23], [[q]]])

same-type?( a, z)

float?( a)

same-structure?( [[b, 3.14], [[c]]], [[x, 1.23], [[q]]])

same-structure?( [b, 3.14], [x, 1.23])

same-type?( b, x)

float?( b)

same-structure?( [3.14], [1.23])

same-type?( 3.14, 1.23)

float?( 3.14)

float?( 1.23)

same-structure?( [], [])

same-structure?( [[[c]]], [[[q]]])

same-structure?( [[c]], [[q]])

same-structure?( [c], [q])

same-type?( c, q)

float?( c)

same-structure?( [], [])

same-structure?( [], [])

same-structure?( [], [])

same-structure?( [], [])

**PROGRAM RESULT >>> #t**

>

(same-structure? '(1 (a (b 3.14) ((c)))) '(3 (z (x 3) ((q)))))

same-type?( 1, 3)

same-structure?( [[a, [b, 3.14], [[c]]]], [[z, [x, 3], [[q]]]])

same-structure?( [a, [b, 3.14], [[c]]], [z, [x, 3], [[q]]])

same-type?( a, z)

float?( a)

same-structure?( [[b, 3.14], [[c]]], [[x, 3], [[q]]])

same-structure?( [b, 3.14], [x, 3])

same-type?( b, x)

float?( b)

same-structure?( [3.14], [3])

same-type?( 3.14, 3)

float?( 3.14)

float?( 3)

**PROGRAM RESULT >>> #f**

>

(same-structure? '(1 2 3 4 5) '(5 4 3 2))

same-type?( 1, 5)

same-structure?( [2, 3, 4, 5], [4, 3, 2])

same-type?( 2, 4)

same-structure?( [3, 4, 5], [3, 2])

same-type?( 3, 3)

same-structure?( [4, 5], [2])

same-type?( 4, 2)

same-structure?( [5], [])

**PROGRAM RESULT >>> #f**

>

(same-structure? '() '())

**PROGRAM RESULT >>> #t**

>

(same-structure? '(("hello") "world") '(("good-bye") "sam"))

same-structure?( ["hello"], ["good-bye"])

same-type?( "hello", "good-bye")

float?( "hello")

same-structure?( [], [])

same-structure?( ["world"], ["sam"])

same-type?( "world", "sam")

float?( "world")

same-structure?( [], [])

PROGRAM RESULT >>> #t

>

(define sandwich-first

(lambda (a b lst)

(cond

((null? lst) '())

((null? (cdr lst)) lst)

((and (equal? b (car lst)) (equal? b (cadr lst)))

(append (list b a b) (cddr lst)))

(else (cons (car lst) (sandwich-first a b (cdr lst)))))

))

>

(sandwich-first 'meat 'bread '(bread bread))

**PROGRAM RESULT >>> [bread, meat, bread]**

>

(sandwich-first 'meat 'bread '())

**PROGRAM RESULT >>> []**

>

(sandwich-first 'meat 'bread '(meat meat))

sandwich-first( meat, bread, [meat])

**PROGRAM RESULT >>> [meat, meat]**

>

(define min-to-head

(lambda (lst)

(cond

((null? lst) '())

((null? (cdr lst)) lst)

(else (let\* ((new-lst (min-to-head (cdr lst)))

(second (car new-lst)))

(if (> (car lst) second)

(cons second (cons (car lst) (cdr new-lst)))

lst)))

)))

>

(min-to-head '(5 8 1 0 6 2 1 9))

min-to-head( [8, 1, 0, 6, 2, 1, 9])

min-to-head( [1, 0, 6, 2, 1, 9])

min-to-head( [0, 6, 2, 1, 9])

min-to-head( [6, 2, 1, 9])

min-to-head( [2, 1, 9])

min-to-head( [1, 9])

min-to-head( [9])

**PROGRAM RESULT >>> [0, 5, 8, 1, 6, 2, 1, 9]**

>

(define sort

(lambda (lst)

(if (null? lst)

'()

(let ((mth (min-to-head lst)))

(cons (car mth) (sort (cdr mth)))))

))

>

(sort '(5 8 1 0 6 2 1 9))

min-to-head( [5, 8, 1, 0, 6, 2, 1, 9])

min-to-head( [8, 1, 0, 6, 2, 1, 9])

min-to-head( [1, 0, 6, 2, 1, 9])

min-to-head( [0, 6, 2, 1, 9])

min-to-head( [6, 2, 1, 9])

min-to-head( [2, 1, 9])

min-to-head( [1, 9])

min-to-head( [9])

sort( [5, 8, 1, 6, 2, 1, 9])

min-to-head( [5, 8, 1, 6, 2, 1, 9])

min-to-head( [8, 1, 6, 2, 1, 9])

min-to-head( [1, 6, 2, 1, 9])

min-to-head( [6, 2, 1, 9])

min-to-head( [2, 1, 9])

min-to-head( [1, 9])

min-to-head( [9])

sort( [5, 8, 6, 2, 1, 9])

min-to-head( [5, 8, 6, 2, 1, 9])

min-to-head( [8, 6, 2, 1, 9])

min-to-head( [6, 2, 1, 9])

min-to-head( [2, 1, 9])

min-to-head( [1, 9])

min-to-head( [9])

sort( [5, 8, 6, 2, 9])

min-to-head( [5, 8, 6, 2, 9])

min-to-head( [8, 6, 2, 9])

min-to-head( [6, 2, 9])

min-to-head( [2, 9])

min-to-head( [9])

sort( [5, 8, 6, 9])

min-to-head( [5, 8, 6, 9])

min-to-head( [8, 6, 9])

min-to-head( [6, 9])

min-to-head( [9])

sort( [8, 6, 9])

min-to-head( [8, 6, 9])

min-to-head( [6, 9])

min-to-head( [9])

sort( [8, 9])

min-to-head( [8, 9])

min-to-head( [9])

sort( [9])

min-to-head( [9])

sort( [])

**PROGRAM RESULT >>> [0, 1, 1, 2, 5, 6, 8, 9]**

>

(define mistake

(lambda (a b a)

(let ((d (+ a b)))

d)))

**a is already defined.**

>

(member? 'a)

**PROGRAM RESULT >>> Incorrect number of argument passed. Expected: 2. Passed: 1**

>