Michael Naughton – 16461226

https://github.com/Raymk111/ct331\_assignment2

Q1:

#lang racket

(cons 1 2)

(cons 1 (cons 2 (cons 3 empty)))

(cons "MUP" (cons 70 (cons (cons 1 (cons 2 (cons 3 empty))) empty)))

(list "MUP" 70 (list 1 2 3 ))

(append '("MUP") '(70) '((1 2 3)))

;PART B:

;cons is for constructing pairs of elements in either a dotted or proper list

;to construct a proper list with cons the last element must be paired with empty or '()

;append on the other hand only takes lists and concatenates their top level elements to one list

;list takes any number of lists and elements and concatenates them all together on a new list one step up

;each element being an element in the list and each list being an element

Q2:

#lang racket

;This is an example implementation of ins\_beg,

;It obviously doesn't do what it should, so you

;can edit this function to get started.

;

;Please note the provide function is necessary

;for the unit tests to work. Please include a

;(provide) for each function you write in your

;submitted assignment.

;

;You may delete these comments!

(provide ins\_beg)

;A.

(define (ins\_beg el1 list1)

(cons el1 list1)

)

(provide ins\_end)

;B.

(define (ins\_end el1 list1)

(append list1 (cons el1 empty))

)

(provide count\_top\_level)

;C.

(define (count\_top\_level list1 length)

(if (empty? list1)

length

(begin

(set! length ((lambda (length) (+ length 1)) length))

(set! list1 ((lambda (list1) (cdr list1)) list1))

(count\_top\_level list1 length )

)

)

)

(provide count\_occurence)

;D.

(define (count\_occurence list1 num)

(cond

[(empty? list1) 0]

[(= num (car list1)) (+ 1 (count\_occurence (cdr list1) num))]

[(not (= num (car list1))) (+ 0 (count\_occurence (cdr list1) num))]

)

)

(provide count\_occurence1)

;E

(define (count\_occurence1 list1 num count)

(cond

[(empty? list1) count]

[(= num (car list1)) (begin

(set! count ((lambda (count) (+ count 1)) count))

(count\_occurence1 (cdr list1) num count)

)

]

[(not (= num (car list1))) (count\_occurence1 (cdr list1) num count)]

)

)

(provide count\_instance\_deep)

;F

(define (count\_instance\_deep list1 num)

(cond

[(empty? list1) 0]

[(and (list? (car list1))) (+ (count\_instance\_deep (car list1) num)

(count\_instance\_deep (cdr list1) num)

)]

[(= num (car list1)) (+ 1

(count\_instance\_deep (cdr list1) num))]

[(not (= num (car list1))) (+ 0 (count\_instance\_deep (cdr list1) num))]

)

)

Q3:

#lang racket

(define bst (list (list(list(list '() 1 '()) 2 (list '() 3 '()))4 '()) 5 (list '() 6 (list (list '() 7 '()) 8 (list '() 9 '())))))

(define (traverse tree)

(cond [(not (empty? (car tree))) (traverse (car tree))])

(cond [(not (empty? (cadr tree))) (display (cadr tree))])

(cond [(not (empty? (caddr tree))) (traverse (caddr tree))])

)

(traverse bst)

(define (contains tree num)

(cond

[(and (not (empty? (car tree))) (< num (cadr tree)))

(contains (car tree) num)]

[(and (not (empty? (cadr tree))) (= num (cadr tree)))

(display #t)]

[(and (not (empty? (caddr tree))) (> num (cadr tree)))

(contains (caddr tree) num)]

[else (display #f)]

)

)

(define (insert tree num)

(cond

[(empty? tree) (list '() num '())]

[(and (not (empty? (cadr tree))) (= num (cadr tree)))

(display "ITEM ALREADY EXISTS")]

[(> num (cadr tree))

(cond

[(empty? (caddr tree)) (reverse (append (list (list '() num '())) (cdr (reverse tree))))]

[else (list (car tree) (cadr tree) (insert (caddr tree) num))]

)

]

[else

(cond

[(empty? (car tree)) (append (list (list '() num '())) (cdr tree))]

[else (list (insert (car tree) num) (cadr tree) (caddr tree))]

)

]

)

)

(define (insert\_list tree lst)

(cond

[(empty? lst) tree]

[(and (not (empty? (car lst))) (not (list? (car lst))))

(insert\_list (insert tree (car lst)) (cdr lst))]

[else (display "error with tree construction list must not be nested")]

)

)

(define (tree\_sort lst)

(traverse (insert\_list empty lst))

)

;F

(define (insert\_comp tree num comparator)

(cond

[(empty? tree) (list '() num '())]

[(and (not (empty? (cadr tree))) (= num (cadr tree)))

(display "ITEM ALREADY EXISTS")]

[(comparator num (cadr tree))

(cond

[(empty? (caddr tree)) (reverse (append (list (list '() num '())) (cdr (reverse tree))))]

[else (list (car tree) (cadr tree) (insert\_comp (caddr tree) num comparator))]

)

]

[else

(cond

[(empty? (car tree)) (append (list (list '() num '())) (cdr tree))]

[else (list (insert\_comp (car tree) num) (cadr tree) (caddr tree) comparator)]

)

]

)

)

(define (insert\_comp\_list tree lst comparator)

(cond

[(empty? lst) tree]

[(and (not (empty? (car lst))) (not (list? (car lst))))

(insert\_comp\_list (insert\_comp tree (car lst) comparator) (cdr lst) comparator)]

[else (display "error with tree construction list must not be nested")]

)

)

Tests:

#lang racket

(require (file "assignment\_q2.rkt")

(file "assignment\_q3.rkt"))

;

;Don't worry about this file unless you are doing the extra credit tests.

;

;This structure allows a single function call

;to run every test in sequence, rather than

;calling each function separately.

(define (runTests)

(begin

(display "Running tests...\n")

;begin calling test functions

(printf "1: ~a\n" (test\_ins\_beg1))

(printf "2: ~a\n" (test\_ins\_end1))

(printf "3: ~a\n" (test\_count\_top\_level))

(printf "4: ~a\n" (test\_count\_occurence))

(printf "5: ~a\n" (test\_count\_occurence1))

(printf "6: ~a" (test\_count\_instance\_deep))

;end calling test functions

(display "\nTests complete!\n"))

)

;Begin test functions

(define (test\_ins\_beg1)

(equal? (ins\_beg 1 '(2 3 4)) '(1 2 3 4)))

(define (test\_ins\_end1)

(equal? (ins\_end 1 '(2 3 4)) '(2 3 4 1)))

(define (test\_count\_top\_level)

(equal? (count\_top\_level (list 1 (list 2 (list 0 5) 3 4)) 0) 5))

(define (test\_count\_occurence)

(equal? (count\_occurence (list 1 2 5 4 5 3 5) 5) 3))

(define (test\_count\_occurence1)

(equal? (count\_occurence1 (list 1 2 3 4 5 3 5) 3 0) 2))

(define (test\_count\_instance\_deep)

(equal? (count\_instance\_deep (list 1 (list 2 (list 0 5) 3 4)) 0) 1))

;End test functions

;Run the tests

(runTests)



