CT331 Assignment 2

Daniel Hannon (19484286)

November 2021

1 Question 1

1.1 Code

```
#lang scheme
;;question1.rkt - Author Daniel Hannon(19484286)
(cons 2 3)
(cons 2 '(3 4))
(cons "string" '((1 2 3)))
(list "string2" '(4 5 6))
(append '("string3") (append '(4) '((7 8 9))))
```

1.2 Output

```
Welcome to DrRacket, version 8.3 [cs].
Language: scheme, with debugging; memory limit: 128 MB.
(2 . 3)
(2 3 4)
("string" (1 2 3))
("string2" (4 5 6))
("string3" 4 (7 8 9))
>
```

1.3 Part B

Cons, when passed two atomics creates a list with the dot to indicate where car returns (the left) and where cdr returns (the right)
If you pass a List as the first or second parameter it no longer features the dot.
In order to feature a list within it, you must encase the list within another set of brackets.
With List you do not need to do this .
With append you cannot pass atomics so everything must be passed as a list.

2 Question 2

2.1 Code

```
#lang scheme
;;question2.rkt - Author Daniel Hannon(19484286)
;;Q2 Part A
(define (ins_beg val_1 val_2)
  ;;I have to define val_1 as a list
  ;;within this or it does not work
  (append (list val_1) val_2))
;;Q2 Part B
(define (ins_end val_1 val_2)
  ;;I have to define val_1 as a list
  ;;within this or it does not work
  (append val_2 (list val_1)))

;;Q2 Part C
(define (cout_top_level x)
  ;;Count the number of top levels
  (if (empty? x)
      0
      (+ 1 (cout_top_level (cdr x)))
      )
  )

;;Q2 Part D
(define (count_instances val list)
  ;;count instances
  (if (empty? list)
      0
      (if (= val (car list))
          (+ 1 (count_instances val (cdr list)))
          (+ 0 (count_instances val (cdr list)))
          )
      )
  )

;;Q2 Part E
;;wrapper for the function
(define (count_instances_tr val list)
  (count_instances_tr_inner val list 0)
)

(define (count_instances_tr_inner val list tally)
  (if (empty? list)
      ;;returns tally if the list is fully traversed
      tally
      (if (= val (car list))
          ;;If it is true the tally is increased by one, otherwise it remains the same
          (count_instances_tr_inner val (cdr list) (+ tally 1))
          (count_instances_tr_inner val (cdr list) tally)
          )
      )
  )

;;Q2 Part F
(define (count_instances_deep val list)
  ;;flatten turns any deep list into a flat one :)
  ;;Its a built in function in scheme
```

```
  (count_instances_tr_inner val (flatten list) 0)
  )

;;Tests
(printf "Testing ins_beg\n")
(ins_beg 'a '(b c d))
(ins_beg '(a b) '(b c d))
(printf "Testing ins_end\n")
(ins_end 'a '(b c d))
(ins_end '(a b)  '(b c d))
(printf "Testing cout_top_level\n")
;; should return 3
(cout_top_level '(a b c))
;; should return 4
(cout_top_level '((a b) c d ((e) f)))
;;should return 1
(cout_top_level '((a(b(c(d))))))

(printf "Testing count_instances\n")
;;Should return 3
(count_instances 7 '(7 7 5 7))
;;Should return 2
(count_instances 2 '(2 0 0 2 0 0))

(printf "Testing count_instances_tr\n")
;;Should return 3
(count_instances_tr 7 '(7 7 5 7))
;;Should return 2
(count_instances_tr 2 '(2 0 0 2 0 0))

(printf "testing count_instances_deep\n")
(count_instances_deep 7 '((1 7) 4 6 ((7) 2)))
(count_instances_deep 7 '((1 7 ( 7 (0 (1 2 (6 7)))) 7)))
```

2.2 Output

```
Welcome to DrRacket, version 8.3 [cs].
Language: scheme, with debugging; memory limit: 128 MB.
Testing ins_beg
(a b c d)
((a b) b c d)
Testing ins_end
(b c d a)
(b c d (a b))
Testing cout_top_level
3
4
1
Testing count_instances
3
2
Testing count_instances_tr
3
2
testing count_instances_deep
2
4
> 
```

3 Question 3

3.1 Code

```
#lang scheme
;;question3.rkt Author - Daniel Hannon (19484286)

;;Q3 Part A
(define (btree_traverse btree)
  (if (empty? btree)
      (printf "")
      (begin
        ;;traverse left
        (btree_traverse (car btree))
        ;;print node value
        (print (cadr btree))
        (printf ", ")
        ;;traverse right
        (btree_traverse (caddr btree))
       )
  )
)

;;Q3 Part B
(define (btree_search val btree)
  (if (empty? btree)
      #f
      (cond
        ;;check if equal to the value stored at the node
        [(= (cadr btree) val) #t]
        ;;Check if less than the value stored at the node,
        ;;as a binary search tree is ordered, only check if it is less than
        [(> (cadr btree) val) (btree_search val (car btree))]
        ;;Check if greater than the value stored at the node
        [(< (cadr btree) val) (btree_search val (caddr btree))]
        )
      )
  )

;;Q3 Part C
(define (btree_insert val btree)
  (if (empty? btree)
      (list '() val '())
      (cond
        ;;IF GEQ node value - Insert Right of node
        ;;IF LT node value - Insert Left of node
        [(<= (cadr btree) val) (list (car btree) (cadr btree) (btree_insert val (caddr
        ↪  btree)))]
        [(> (cadr btree) val) (list (btree_insert val (car btree)) (cadr btree) (caddr
        ↪  btree))]
        )
      )
  )

;:Q3 Part D

(define (list_to_btree list)
  (list_to_btree_inner list '())
  )
```

```
(define (list_to_btree_inner list btree)
  (if (empty? list)
      btree
      (list_to_btree_inner (cdr list) (btree_insert (car list) btree))
      )
  )

;;Q3 Part E
(define (tree_sort_list list)
  ;;Technically The solution to D does this
  (btree_traverse (list_to_btree list))
  )


;;Discriminator functions for Q3 Part F

;;Sort in Ascending order
(define (ascending_sort x y)
  (cond
    [(= x y) 0]
    [(> x y) 1]
    [(< x y) -1]
    )
  )

;;Sort in Descending order
(define (descending_sort x y)
  (ascending_sort y x)
  )

;;Ascending Least Significant Digit
(define (ascending_lsd x y)
  (ascending_sort (modulo x 10) (modulo y 10))
  )

;;Higher Order Binary Tree Insert Function as per Q3 Part F
(define (higher_order_btree_insert val btree discrim)
  (if (empty? btree)
      (list '() val '())
      (cond
        ;;Evaluate Discriminator and act in accordance to it
        ;;GEQ insert to the right
        ;;LT insert to the left
        [(<= (discrim (cadr btree) val) 0)
         (list (car btree) (cadr btree) (higher_order_btree_insert val (caddr btree)
          ↪ discrim))
         ]
        [(> (discrim (cadr btree) val) 0)
         (list (higher_order_btree_insert val (car btree) discrim) (cadr btree) (caddr
          ↪ btree))
         ]
        )
      )
  )

;;Companion Functions for Q3 Part F for QOL while testing
(define (list_to_btree_higher_order list discrim)
  (list_to_btree_higher_order_inner list '() discrim)
  )
```

```
(define (list_to_btree_higher_order_inner list btree discrim)
  (if (empty? list)
      btree
      (list_to_btree_higher_order_inner (cdr list) (higher_order_btree_insert (car list)
      ↪  btree discrim) discrim)
      )
  )

;;Tests
;;Part A
(printf "Testing btree traverse\n")
(btree_traverse '((() 1 ()) 2 (() 3 ())) 4 ((() 5 ()) 6 (() 7 ())))
;;Part B
(printf "\ntesting btree search\n")
;;Return True
(btree_search '4 '((() 1 ()) 2 (() 3 ())) 4 ((() 5 ()) 6 (() 7 ())))
;:Return False
(btree_search '8 '((() 1 ()) 2 (() 3 ())) 4 ((() 5 ()) 6 (() 7 ())))
;;Part C
(printf "testing btree_insert\n")
(btree_traverse (btree_insert '8 '((() 1 ()) 2 (() 3 ())) 4 ((() 5 ()) 6 (() 7 ()))))
;;Part D
(printf "\ntesting creating a btree from a list\n")
(btree_traverse (list_to_btree '(4 6 7 3 5 2 4 5 11 23 45 67 13 9 1 0)))
;;Part E
(printf "\nTesting Sorting a list using a binary tree\n")
(tree_sort_list '(9 8 7 6 5 4 3 2 1))
;;Part F
(printf "\nTesting Higher Order btree\n")
(printf "Ascending order\n")
(btree_traverse (list_to_btree_higher_order '(4 3 5 1 2 6 7) ascending_sort))
(printf "\nDescending order\n")
(btree_traverse (list_to_btree_higher_order '(4 3 5 1 2 6 7) descending_sort))
(printf "\nAscending Least Significant Digit\n")
(btree_traverse (list_to_btree_higher_order '(12 72 63 85 94 21 9) ascending_lsd))
```

3.2 Output

```
Welcome to DrRacket, version 8.3 [cs].
Language: scheme, with debugging; memory limit: 128 MB.
Testing btree traverse
1, 2, 3, 4, 5, 6, 7,
testing btree search
#t
#f
testing btree_insert
1, 2, 3, 4, 5, 6, 7, 8,
testing creating a btree from a list
0, 1, 2, 3, 4, 4, 5, 5, 6, 7, 9, 11, 13, 23, 45, 67,
Testing Sorting a list using a binary tree
1, 2, 3, 4, 5, 6, 7, 8, 9,
Testing Higher Order btree
Ascending order
1, 2, 3, 4, 5, 6, 7,
Descending order
7, 6, 5, 4, 3, 2, 1,
Ascending Least Significant Digit
21, 12, 72, 63, 94, 85, 9,
>
```