CT331 Assignment 2

Programming Paradigms

Maxwell Maia 21236277

Question 1

(A)

```
#lang racket

(cons 1 2)

(cons 1 (cons 2 (cons 3 '())))

(cons "My epic string"
      (cons 164
            (cons (cons 1 (cons 2 (cons 3 '())))
                  '())))

(list "hi" 2 (list 1 2 3))

(append '("Hello") '(12) '(1 2 3))
```

Output

```
'(1 . 2)
'(1 2 3)
'("My epic string" 164 (1 2 3))
'("hi" 2 (1 2 3))
'("Hello" 12 1 2 3)
```

(B)

Cons creates cons pairs.

You can create linked lists by adding elements to the front of an existing list.

To create a list with the cons function you need to cons a number to the an “empty”. You can extend that list by cons’ing more elements onto that list.

List creates new lists easily, you just need to put spaces in between the elements. List can create lists with lists in them.

Append combines existing lists into a single, longer list.

Question 2

```
#lang racket


;A.
(provide ins_beg)


(define (ins_beg el lst)
  (cons el lst))




;B.
(provide ins_end)


(define (ins_end el lst)
  (append lst (list el)))




;C.
;returns the number of top-level elements in a list
```

```
(provide cout_top_level)

(define (cout_top_level lst)
  (if (null? lst)
      0
      (+ 1 (cout_top_level (cdr lst)))
  )
)
; example code: (cout_top_level '(2 3 4 6 6 '(2 2))) should return 6




;D.
;non-tail recursive function that counts the times an item occurs in a list of items
(provide count_instances)

(define (count_instances item lst)
  (if (null? lst)
      0
      (if (equal? item (car lst))
          (+ 1 (count_instances item (cdr lst)))
          (count_instances item (cdr lst))
      )
  )
)
;example code: (count_instances 1 '(1 2 3 1 4 5 1 6)) should return 3
```

```
;E.
;tail recursive function, counts items in a list of items
(provide count_instances_tr)

(define (count_instances_tr item lst)
  (count-helper lst item 0)
)

(define (count-helper lst item count)
  (cond
    ((null? lst) count)
    ((equal? item (car lst)) (count-helper (cdr lst) item (+ 1 count)))
    (else (count-helper (cdr lst) item count)))
)
;example code: (count_instances_tr 1 '(1 2 3 1 4 5 1 6)) should return 3




;F.
;returns the instances of an item in a list, also checking sub-lists in the list
(provide count_instances_deep)

(define (count_instances_deep el lst)
  (if (null? lst)
      0
      (if (list? (car lst))
          (+ (count_instances_deep el (car lst)) (count_instances_deep el (cdr lst)))
           (if (= el (car lst))
               (+ 1 (count_instances_deep el (cdr lst)))
               (count_instances_deep el (cdr lst))
```

```
            )
        )
    )
)
; example code: (count_instances_deep 6 '((1 6 6) 6 7 2 5)) should return 3
```

Question 3

```
;D.
;Insert a list of items into a binary search tree.
(provide make-node)
(provide value)
(provide left-subtree)
(provide right-subtree)

(define (make-node value left right)
  (list 'node value left right))

(define (value node)
  (if (null? node) #f
      (cadr node)))

(define (left-subtree node)
  (if (null? node) #f
      (caddr node)))

(define (right-subtree node)
  (if (null? node) #f
```

```
      (cadddr node)))

;C.
;Insert an item into a list representing a binary search tree.
(provide insert-bst)
(define (insert-bst tree element comp-func)
  (cond
    ((null? tree) (make-node element '() '()))
    ((comp-func element (value tree)) (make-node (value tree)
                                       (insert-bst (left-subtree tree) element comp-func)
                                       (right-subtree tree)))
    (else (make-node (value tree)
                     (left-subtree tree)
                     (insert-bst (right-subtree tree) element comp-func)))))

;D.
;Take a list of items and insert them into a binary search tree.
(provide list-to-BST)
(define (list-to-BST lst)
  (define (insert-elements tree elements)
    (if (null? elements)
        tree
        (insert-elements (insert-bst tree (car elements) ascending) (cdr elements)))
    )
  (insert-elements '() lst))

;A.
;In-order traversal to display contents of bst in sorted order
(provide in-order-traversal)
(define (in-order-traversal tree)
```

```
  (if (null? tree)
      '()
      (append (in-order-traversal (left-subtree tree))
              (list (value tree))
              (in-order-traversal (right-subtree tree)))))

;B.
;Returns #t if an item is present in a tree. #f if not
(provide present-in-tree)
(define (present-in-tree item tree)
  (cond
    ((null? tree) #f) ; Base case. Empty tree - item not found
    ((equal? item (car tree)) #t) ; Item found at the root
    (else (or (present-in-tree item (cadr tree)) ; Search left subtree
              (present-in-tree item (caddr tree))))) ; Search right subtree

;E.
;Tree sort function
(provide tree-sort)
(define (tree-sort lst)
  (in-order-traversal (list-to-BST lst)))

;F.
; Extended tree sort function
(provide tree-sort-extended)
(define (tree-sort-extended lst comp-func)
  (define (insert-elements tree elements)
    (if (null? elements)
        tree
        (insert-elements (insert-bst tree (car elements) comp-func) (cdr elements)))
```

```
    )
  (in-order-traversal (insert-elements '() lst)))

;Comparison functions for F
(provide ascending)
(provide descending)
(provide ascending-based-on-last-digit)

(define (ascending a b)
  (< a b))

(define (descending a b)
  (> a b))

(define (ascending-based-on-last-digit a b)
  (< (modulo a 10) (modulo b 10)))

;example code:
;create example list
(define my-list '(5 2 7 4 2 4 6))

;create BST example
(define my-bst (list-to-BST my-list))

;A example
(display "In order traversal of the BST: ")
(display (in-order-traversal my-bst))
(newline)

;E example
```

```
(display "Sorted using function: ")
(display (tree-sort my-list))
(newline)

;F example
(define my-list-two '(5 32 8 42 23 12 51))

(newline)
(display "Tree sort that uses a comparison function (sort in any order you choose)")
(newline)
(display "Sorting the list (5 32 8 42 23 12 51) into...")
(newline)
(display "Ascending order: ")
(display (tree-sort-extended my-list-two ascending))

(newline)
(display "Descending order: ")
(display (tree-sort-extended my-list-two descending))

(newline)
(display "Ascending order based on last digit: ")
(display (tree-sort-extended my-list-two ascending-based-on-last-digit))
```

Output:

```
In order traversal of the BST: (2 2 4 4 5 6 7)
Sorted using function: (2 2 4 4 5 6 7)

Tree sort that uses a comparison function (sort in any order you choose)
Sorting the list (5 32 8 42 23 12 51) into...
```

Ascending order: (5 8 12 23 32 42 51)

Descending order: (51 42 32 23 12 8 5)

Ascending order based on last digit: (51 32 42 12 23 5 8)

```
Welcome to DrRacket, version 8.10 [cs].
Language: racket, with debugging; memory limit: 128 MB.
In order traversal of the BST: (2 2 4 4 5 6 7)
Sorted using function: (2 2 4 4 5 6 7)

Tree sort that uses a comparison function (sort in any order you choose)
Sorting the list (5 32 8 42 23 12 51) into...
Ascending order: (5 8 12 23 32 42 51)
Descending order: (51 42 32 23 12 8 5)
Ascending order based on last digit: (51 32 42 12 23 5 8)
>
```