CT331 Assignment 2
Raphaël Marzat 18101202

https://github.com/MrWazzat/ct331_assignment2

Question 1 :
Code :

```
#lang racket
;Cons pair of 2 numbers
(cons 1 2)
;List of 3 numbers built with the cons function
(cons 3 (cons 4 (cons 5 empty)))
;List containing a String, a number and a nested list of three numbers (cons function)
(cons "cons" (cons 6 (cons '(7 8 9) empty)))
;List containing a String, a number and a nested list of three numbers (list function)
(list "list" 10 '(11 12 13))
;List containing a String, a number and a nested list of three numbers (append function)
(append '("append") '(14) '((15 16 17)))
```

Output :

```
'(1 . 2)
'(3 4 5)
'("cons" 6 (7 8 9))
'("list" 10 (11 12 13))
'("append" 14 (15 16 17))
>
```

Comment :

The results are always the same. Nevertheless, when using cons, we have to "pair" everything, and if we don't want the dot before the last element, we have to pair it with an empty element.

When using the list function, everything is simpler, because we just have to pass the elements we want as arguments and it builds a list with them.

When using the append function, we have to pass all the elements as lists (because append only takes lists parameter). Thus if we want the last element to be a nested list, we have to make a list where the only element is the nested list.

Question 2:
Code :

```
#lang racket

(provide ins_beg)
(provide ins_end)
(provide cout_top_level)
(provide count_instances)
(provide count_instances_tr)
(provide count_instances_tr_helper)
(provide count_instances_deep)

;;Atom function for the deep search
(define (atom? x)
  (not (or (pair? x) (null? x)))
)

(define (ins_beg el lst)
  ;The (cons) function is used here to turn el into a list so it can be used in (append)
  (append (cons el empty) lst)
)

(define (ins_end el lst)
  ;The (cons) function is used here to turn el into a list so it can be used in (append)
  (append lst (cons el empty))
)

(define (cout_top_level list)
  (if (not (empty? list))
     (+ 1 (cout_top_level (cdr list)))
  0)
)


(define (count_instances elem list)
  (cond [(empty? list) 0]
        [(= elem (car list)) (+ 1 (count_instances elem (cdr list)))]
        [else(count_instances elem (cdr list))]
  )
)

(define (count_instances_tr elem list)
  (count_instances_tr_helper elem list 0)
)


(define (count_instances_tr_helper elem list number)
  (cond [(empty? list) number]
        [(= elem (car list)) (count_instances_tr_helper elem (cdr list)(+ 1 number))]
        [else (count_instances_tr_helper elem (cdr list) number)]
   )
)

(define (count_instances_deep elem list)
  ;;if the list is empty return 0
  (cond[(empty? list) 0]
       ;;if the first element isn't an atom, we count the number of elements inside of it
       [(not(atom? (car list))) (+ (count_instances_deep elem (car list)) (count_instances_deep elem (cdr list)))]
       ;;if the first element is the good 1 we add 1
       [(= elem (car list)) (+ 1 (count_instances_deep elem (cdr list)))]
       ;;if it isn't we go to the next element
       [else (count_instances_deep elem (cdr list))]
))
```

```
;Basic tests
(display "Tests ins_beg \n")
(ins_beg 'a '(b c d))
(ins_beg '(a b) '(b c d))

(display "\nTests ins_end \n")
(ins_end 'a '(b c d))
(ins_end '(a b) '(b c d))

(display "\nTests cout_top_level \n")
(cout_top_level '(b c d))
(cout_top_level '(b "Hello"))
(cout_top_level '(b "Hello" '("sublist" 1 a b) 5))

(display "\nTests count_instances \n")
(count_instances 5 '(5 2 3 6 5 4 5))
(count_instances 5 '(5))
(count_instances 5 '())
(count_instances 5 '(5 5 5 5 5 5))
(count_instances 5 '(2 1 3 6 8 7))

(display "\nTests count_instances_tr \n")
(count_instances_tr 5 '(5 2 3 6 5 4 5))
(count_instances_tr 5 '(5))
(count_instances_tr 5 '())
(count_instances_tr 5 '(5 5 5 5 5 5))
(count_instances_tr 5 '(2 1 3 6 8 7))
```

```
(display "\nTests count_instances_deep \n")
(count_instances_deep 5 '(5 (2 3 6 5) 4 5))
(count_instances_deep 5 '(5))
(count_instances_deep 5 '())
(count_instances_deep 5 '(5 (5 5 5) 5 5))
(count_instances_deep 5 '(2 (5 3 (5 (5) 2) 1) 3 6 8 7))
```

Output question 2 :

```
Tests ins_beg
'(a b c d)
'((a b) b c d)

Tests ins_end
'(b c d a)
'(b c d (a b))

Tests cout_top_level
3
2
4

Tests count_instances
3
1
0
6
0

Tests count_instances_tr
3
1
0
6
0

Tests count_instances_deep
3
1
0
6
3
>
```

Question3 :

Code :

```
#lang racket

;;Using the definition from the slides :
;;Left child is the first element in the list
;;Value is second element in the list
;;Right is the last element in the list

(provide display_sorted_bst)
(provide search_elem_bst)
(provide insert_elem_bst)
;;Atom function
(define (atom? x)
  (not (or (pair? x) (null? x)))
)

;;Displays the bst passed as an argument in sorted order
(define (display_sorted_bst bst)
  (cond[(empty? bst)]
       [(atom? (car bst)) (display (car bst))
                          (display " ")
                          (display_sorted_bst (cdr bst))]
       [else (display_sorted_bst (car bst))
             (display_sorted_bst (cdr bst))]
   )
)

;;Searches an element in a bst
(define (search_elem_bst elem bst)
  (cond [(empty? bst) #f]
        [(atom? bst)(if(= elem bst)#t #f)]
        [(= elem (cadr bst))]
        [(< elem (cadr bst)) (search_elem_bst elem (car bst)) ]
        [else (search_elem_bst elem (caddr bst))]
  )
)

(define (insert_elem_bst elem bst)
  (cond[(empty? bst) (list bst elem)]
       [(atom? bst)(if(< elem bst)(list elem bst '()) (list '() bst elem))]
       [(= elem (cadr bst)) (display "The element is already in the tree")]
       [(< elem (cadr bst)) (insert_elem_bst elem (car bst)) ]
       [else (insert_elem_bst elem (caddr bst))]
  )
)
```

```
(display_sorted_bst '((1 3(4 6 7))8(() 10 (13 14 ()))))

(search_elem_bst 6'((1 3(4 6 7))8(() 10 (13 14 ()))))
(search_elem_bst 13 '((1 3(4 6 7))8(() 10 (13 14 ()))))
(search_elem_bst 2 '((1 3(4 6 7))8(() 10 (13 14 ()))))
(search_elem_bst 1 '((1 3(4 6 7))8(() 10 (13 14 ()))))
(search_elem_bst 14 '((1 3(4 6 7))8(() 10 (13 14 ()))))

;;Don't know how to keep trace of the tree
(insert_elem_bst 2 '((1 3(4 6 7))8(() 10 (13 14 ()))))
(insert_elem_bst 9 '((1 3(4 6 7))8(() 10 (13 14 ()))))
```

Output :

```
1 3 4 6 7 8 10 13 14 #t
#t
#t
#f
#t
#t
'(() 1 2)
'(() 9)
>
```