Solution for Assignment 2:

Lisp & C programming

COMP-348

by

Shadi Jiha (40131284) and Thomas Gillard (40031490)

Concordia University

Department of Computer Science and Software engineering

08 June 2021

Question 1:

```
(defun sub-list (lst from &optional to)
	(if(null to)
		(cond
			((null lst) nil)
			((> from (list-length lst)) nil)
			((> from 1) (sub-list (cdr lst) (- from 1)))
			((= from 1) (cons(car lst)(sub-list(cdr lst) 1)))
		)

		(cond
			((null lst) nil)
			((> from (list-length lst)) nil)
			((> from to) nil)
			((> from 1) (sub-list (cdr lst) (- from 1) (- to 1)))
			((and(= from 1)(> to 1)) (cons(car lst)(sub-list(cdr lst) 1 (- to 1))))
			((and(= from 1)(= to 1)) (list(car lst)))
		)
	)
)
```

(print(sub-list'(a b c d) 1 3))

(A B C)

Question 2:

```
(defun func(lst)
	(let((a (func-flat lst)))
	(print (func-dups a))
	)
)


(defun func-flat(lst)
	(cond
	((null lst) nil)
	((atom(car lst))(cons(car lst)(func-flat(cdr lst))))
	(t(append (func-flat(car lst))(func-flat(cdr lst))))
```

```
        )
)

(defun func-dups(lst)
        (cond
        ((null lst) nil)
        ((member (car lst)(cdr lst)) (func-dups(cdr lst)))
        (t(cons(car lst) (func-dups(cdr lst))))
        )

)
```

(func '(1 2 (3 1) (a 2.5) (2 4.5) ((1 2))))

(3 A 2.5 4.5 1 2)

Question 3:

```
(defun dept (arg)
        (cond
        ((atom arg) 0)
        (t (max (+ 1 (dept(car arg)))
                (dept (cdr arg))))
        )
)
```

(print(dept '((2)(3 (6))(4))))

3

Question 4:

```
(defun lucas (n &optional (a 2) (b 1))
  (if (zerop n)
        nil
        (cons a (lucas (- n 1) b (+ a b)))))
```

(print(lucas 5))

(2 1 3 4 7)

Question 5:

```
(defun num-positions (lst)
        (if(null lst)
        0
        (+ 1 (num-positions(cdr lst)))
        )
)

(defun num-elements(lst)
        (if (not(null lst))
        (+ (if (atom(car lst))
                1
                (num-elements(car lst))
        )
        (num-elements(cdr lst))
        )
        0
        )
)

(defun calc-num(lst pos &optional (a 1))
        (cond
        ((null lst) 0)
        ((atom(car lst)) (+(* 1 (- a (/ (+ pos 1) 2)))(calc-num(cdr lst) pos (+ a 1))))
        (t (+(* (num-elements(car lst)) (- a (/ (+ pos 1) 2)))(calc-num(cdr lst) pos (+ a 1))))
        )
)

(defun cog(lst)

        (setf pos (num-positions lst))
        (setf elts (num-elements lst))
        (print(float (/ (calc-num lst pos) elts)))

)
```

(cog '(a (b c) d (e (f g))))

0.357

Question 6

```
(defun node-el(node)

        (car node)
)

(defun node-left(node)
        (car (cdr node))
)

(defun node-right(node)
        (car (cdr (cdr node)))
)

(defun is-leaf(node)
        (if (and(null (node-left node))(null (node-right node)))
        t
        nil
        )
)

(defun is-bst-util(root ma mi)

        (if(or(null root))
        t
        (if(or(< (node-el root) mi)(> (node-el root) ma))
        nil
        (if(and(is-bst-util (node-left root) (- (node-el root) 1) mi)
                (is-bst-util (node-right root) ma (+ 1 (node-el root )))
                ) t nil)
        )
        )

)

(defun is-bst(root)
        (print(is-bst-util root 100000 -100000))
)
```

(is-bst '(9 (12 () ())(11 () ())))

Nil

Question 7:

```
(defun node-el(node)

	(car node)
)

(defun node-left(node)
	(car (cdr node))
)

(defun node-right(node)
	(car (cdr (cdr node)))
)

(defun is-leaf(node)
	(if (and(null (node-left node))(null (node-right node)))
	t
	nil
	)
)

(defun inorder-util(root)

	(if(null root)
	nil
	(append (inorder-util (node-left root)) (list(node-el root))(inorder-util (node-right
root)))
	)

)

(defun inorder(root)
	(print(inorder-util root))
)
```

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

(1 3 4 6 7 8 10 13 14)

Question 9:

Here is the implementation of the findmin function:

```
int* findmin(int* arr, int size) {

    int min = arr[0];
      int index = 0;
      for(int i = 0; i < size; i++) {
            if (arr[i] < min) {
                  min = arr[i];
                  index = i;
            }
      }

      return &arr[index];
}
```

Question 10:

Please view source file Ass 2/question 10

Question 11:

Please view source file Ass 2/question 11_12

Question 12:

Please view source file Ass 2/question 11_12

Question 13:

Please view source file Ass 2/question 13