CS 580
Fall 2008
Dr. Duric

Homework #1 Solution

1. (4p)

(a) takes a positive integer and prints that many dots.

Solution:

Recursive:

```
(defun rdot(x)
 (format t ".")
 (if (> x 1)
    (rdot (- x 1))))
```

Iterative:

```
(defun idot(x)
 (do ((i 0 (+ i 1)))
    (( eql i x) (format t "~%"))
     (format t ".")))

   OR
(defun idot(x)
(do ((i x (- i 1)))
    ((< i 1))
    (format t ".")))
```

(b) takes a list and returns the number of times the symbol a occurs in it.

Recursive:

```
(defun ntimes(lst a)
 (if (null lst)
    0
    (if (eql a(car lst))
  (+ 1 (ntimes (cdr lst) a))
       (+ 0 (ntimes (cdr lst) a)))))
```

Iterative:

```
(defun ntimes (lst a)
  (do ((z 0) (l lst (cdr l)))
     ((null l) z)
    (if (eql a (car l))
 (setf z (+ 1 z)))
   z))
```

2. (3p)

(a)
```
(defun summit (lst)
      (remove nil lst)
            (apply #'+ lst))
```

Solution:

Remove returns the list with nil's, ithe list is not updated so we need to use setf

```
(defun summit (lst)
   (setf lst (remove nil lst))
   (apply #'+ lst))
```

OR

```
(defun summit (lst)
   (apply #'+ (remove nil lst)))
```

(b)
```
(defun summit (lst)
      (let ((x (car lst)))
            (if (null x)
                  (summit (cdr lst))
                  (+ x (summit (cdr lst))))))
```

Solution:

It never ends because it never checks if lst is null.

```
(defun summit (lst)
   (if (null lst)
      0
   (let ((x (car lst)))
     (if (null x)
  (summit (cdr lst))
  (+ x (summit (cdr lst)))))))
```

OR

```
(defun summit (lst)
   (let ((lst (remove nil lst)))
      (let ((x (car lst)))
         (if (null x)
            0
            (+ x (summit (cdr lst)))))))))
```

3. (3p)

a)
```
(defun pos+recursive (n lst)
(if (null lst)
nil
(let ( (head-of-list (car lst))
(rest-of-list (cdr lst)))
(cons (+ n head-of-list) (pos+recursive (+ n 1) rest-of-list)))))
(defun pos+ (lst)
(pos+recursive 0 lst))
```

b) **Iteration** –

```
(defun pos++ (lst)
(let ((current-index 0)
(new-list '()))
(dolist (value lst)
(progn
(setf new-list (append new-list (list (+ value current-index))))
(setf current-index (+ current-index 1))))
new-list))
```

c) **Using mapcar**

```
(defun pos+++ (lst)
(let ((current-index -1))
(mapcar #'(lambda (x)
( + x (setf current-index (+ 1 current-index)))) lst)))
```

4. (3p)

```
 (let ((max 0))
  (defun f (x)
    (cond ((< max x) (setq max x))
                (t max))))
```

5. **Tree traversals** (6p):

```
  1) (defun inorder-nested (tree)
  (cond ((null tree) '())
              ((listp tree)
               (append (inorder-nested (second tree))
                           (append (inorder-nested (first tree))
                                       (inorder-nested (third tree)))))
              (t (cons tree '()))))

2) (defun preorder-nested (tree)
  (cond ((null tree) '())
              ((listp tree)
               (append (preorder-nested (first tree))
                           (append (preorder-nested (second tree))
                                       (preorder-nested (third tree)))))
              (t (cons tree '()))))
3) (defun postorder-nested (tree)
  (cond ((null tree) '())
              ((listp tree)
               (append (postorder-nested (second tree))
                           (append (postorder-nested (third tree))
                                       (postorder-nested (first tree)))))
              (t (cons tree '()))))
```

6**. Dictionary** (3p):

Iterative:

```
(defun findall (el dict)
  (let ((l nil))
    (dolist (e dict l)
      (if  (compare el e) (setf l (cons e l))))))

(defun compare (x x1) (equal x (subseq x1 0 (length x))))
```

Recursive:

```
(defun lookup (symbol dict)
  (cond ((null dict) '())
              ((equal symbol (subseq (car dict) 0 (length symbol)))
               (cons (car dict) (lookup symbol (cdr dict))))
              (t
               (lookup symbol (cdr dict)))))
```

7) **Occurrences**

```
(defun occurrences (lst)
  (setq alist '())
  (dolist (x lst)
    (if (not (assoc x alist))
       (push (list x (count x lst)) alist)))
  (sort alist #'> :key #'second))
```