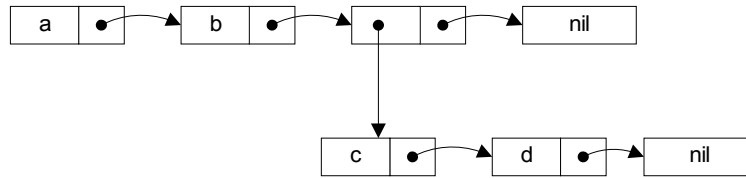


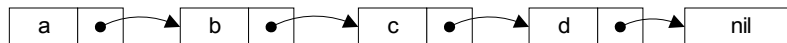
# ANSI Common LISP Chapter 3 Exercises Answers

## 1) Show the following in Box notation

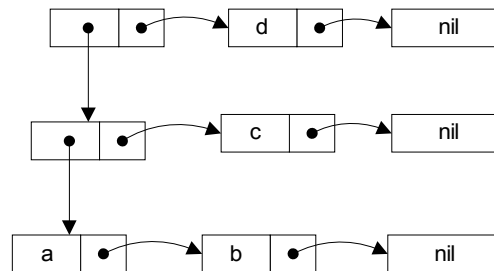
a)



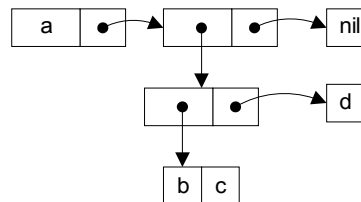
b)



c)



d)



## 2) A version of Union that preserves the order of the elements in the original lists:

```
;;Question 2
(defun new-union2 (lsta lstb)
  (if (null lstb)
      lsta
      (let ( (headb (car lstb))
             (tailb (cdr lstb)))
        (if (null (member headb lsta))
            (new-union2 (append lsta (list headb)) tailb)
            (new-union2 lsta tailb))))))
```

**3) Define a function that takes a list and returns a list indicating the number of times each (eq) element appears, sorted from most common element to least common element.**

```
;;Question 3
(defun inc-letter-count (look-up-table letter)
  (let ((letter-count (cdr (assoc letter look-up-table))))
    (setf (cdr (assoc letter look-up-table)) (+ letter-count 1))
    look-up-table))

(defun occurrence-helper (look-up-table lst)
  (if (null lst)
      look-up-table
      (let ((ahead (car lst))
            (atail (cdr lst)))
        (let ((letter-pair (assoc ahead look-up-table)))
          (if (null letter-pair)
              (occurrence-helper (append look-up-table (list (cons ahead
1))) atail)
              (occurrence-helper (inc-letter-count look-up-table ahead)
atail))))))

(defun occurrence (lst)
  (sort (occurrence-helper '() lst) #'(lambda (a b)
                                       (<= (cdr a) (cdr b)))))
```

*This could be written using mapcar, but I'm too lazy to do that*

**4) Why does (member '(a) '((a)(b))) returns nil?**

*Because the lisp object identified by '(a) is a different lisp object from any member in the list '((a) (b)). Hence, member will assume that the predicate of equality is not being satisfied, and return nil.*

**5) Suppose the function pos+ takes a list and returns a list of each element plus its position**

```
> (pos + '(7 5 1 4))
```

```
(7 6 3 7)
```

**Define this function using (a) recursion (b) iteration © mapcar**

```
;;Question (5a) Recursive version
```

```
(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))
```

;;Question (5b) Iteration - boring

```
(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))
```

;;Question (5c) Using mapcar - interesting

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

## 6) ....

I'm kind of bored with this question, I'm not going to bother with this one.

## 7) **Modify the the program for run-length encoding given the book to use fewer cons cells**

*I can't figure this one out. I'll try this one later.*

## 8) **Define a function that takes a list and prints it in dot notation.**

```
;;Question (8) Defining a function that takes a list and prints it
;;in dot notation
;; This function could probably be written in a more elegant style,
;; but I'm not that conversant with lisp
```

```
(defun showdots (lst)
  (if (atom lst)
```

```
nil
(let ((elt-num 0))
  (mapcar #'(lambda (x)
             (format t "(~A . " x)(setf elt-num (+ elt-num 1))) lst)
  (format t "NIL")
  (do ((i 0 (+ i 1))) ((> i (- elt-num 1)) nil)
      (format t ")"))))
```

**9) Write a program to find the longest finite path through a network represented in the book. The network may contain cycles**

*I figured out a way to do this in an imperative style, but again I couldn't modify the algorithm that's given in the book. I'm going to come back to this question later, when I know some more lisp.*