

Homework:Lisp Exercises 2/Hmwk 3

Now Due Tuesday September 15, 2009 in class

Be sure to follow the guidelines for Programming Assignments. Since these problems are simple, you may skip the Statement of the Problem, Approach and Algorithm - Program Flow sections and give an overall conclusion for the entire programming assignment. Make sure you give me a listing of each program and at least three test cases for each. You may wish to use the "dribble" option in the file menu of XLISP to capture the screen as you test your functions.

- I. Define MYLIST to what the primitive function LIST does.  
(defun mylist (&rest lis) lis)
- II. Define MYAPPEND to what the built-in function APPEND does.  
(See Lisp Notes 1)
- III. Define MYLAST to what the built-in function LAST does.  
(See Lisp Notes 1)
- IV. Write a function, UNNEST, which takes a single list as input and returns a simple list of all its atoms.

Ex. (unnest '(1 (2 (3)) (4 5))) ==> (1 2 3 4 5)  
(unnest '((((test two)))))) ==> (test two)

```
> (defun unnest(lis) (cond
  ((null lis) lis)
  ( (atom (car lis)) (cons (car lis) (unnest (cdr lis))))
  ( t (append (unnest (car lis)) (unnest (cdr lis))))))
```

--or--

```
(defun unnest (lis) (cond
  ((null lis) nil)
  ((atom lis) (list lis))
  (t (append (unnest (car lis)) (unnest (cdr lis)) ) ) )
```

UNNEST

```
> (unnest '(a b c))
(A B C)
> (unnest '((a) ((b)) (c d) (((e))))))
(A B C D E)
> (unnest nil)
NIL
>
```

- V. Write a function, LEN, which is the same as LENGTH. DO NOT use LENGTH in your function definition. Count only top level SEXEs. Make sure that NIL is counted as an atom.

Ex. (len ()) ==> 0  
(len '( a b (c) ((d)) ( ) )) ==> 5  
(len '( ((a ((b))) ))) ==> 1

```
> (defun len(lis) (cond
      ((null lis) 0)
      (t (+ 1 (len (cdr lis))))))
```

LEN

```
> (len '(a b c))
3
> (len nil)
0
> (len '(a))
1
> (len '((a) b (((c))))))
3
> (len '( a b (c) ((d)) ( ) ))
5
> (len '( ((a ((b)) )) ))
1
```

VI. Rewrite union so that the objects remain in order. Call it UNION1.

```
Ex.   (union1 '(a b c) '(f g c d b)) ==> (a b c f g d)
      (union1 '(it is a) '(big box)) ==> (it is a big box)
(defun union1 (s1 s2) (cond
      ((null s1) s2)
      ((member (car s1) s2) (union1 s1 (remove (car s1) s2)))
      (t (cons (car s1) (union1 (cdr s1) s2)))))
```

UNION1

```
> (union1 '(a b) '(c d))
(A B C D)
> (union1 '(a b c) '(f g c d b))
(A B C F G D)
> (union1 '(it is a) '(big box))
(IT IS A BIG BOX)
```

VII. Write a function, SETDIFF, which when given two sets as input, returns all elements in the first set which are not in the second set.

```
Ex.   (setdiff '(a b c d) '(b a) )    ==> (c d)
      (setdiff '(a b c d) '(d e f) )  ==> (a b c)
      (setdiff '(a b c) '(d e f) )    ==> (a b c)
(defun setdiff(s1 s2)(cond
      ((null s1) nil)
      ((member (car s1) s2) (setdiff (cdr s1) s2))
      (t (cons (car s1) (setdiff (cdr s1) s2))) ))
```

SETDIFF

```
> (setdiff '(a b c d) '(b a) )
(C D)
```

```
> (setdiff '(a b c d) '(d e f) )  
(A B C)  
> (setdiff '(a b c) '(d e f) )  
(A B C)
```

VIII. Write a predicate function, EQSET, which returns true if its two arguments are the same set.

```
Ex. (eqset '(this is the way) '(it is this way) ) ==> nil  
    (eqset '(Coke is it) '(it is Coke) )      ==> t  
(defun eqset(s1 s2) (equal (setdiff s1 s2) (setdiff s2 s1)))  
EQSET  
> (eqset '(this is the way) '(it is this way) )  
NIL  
> (eqset '(Coke is it) '(it is Coke) )  
T
```

Also

```
(defun eqset (s1 s2)(cond  
  ((null s1) (null s2))  
  ((null s2) nil)  
  ((and (member (car s1) s2) (member (car s2) s1))  
   (eqset (remove (car s2)(remove (car s1) s1))  
          (remove (car s1) (remove(car s2) s2))))  
  (t nil)))
```

IX. Write a function, MAKESET, which takes a simple list as input and makes a set.

```
Ex. (makeset '(a b a d e b f) ) ==> (a d e b f)  
    (makeset '(1 1 1 2 2 2 3 3) ) ==> (1 2 3)  
(defun makeset(lis)(cond  
  ((null lis) nil)  
  ((member (car lis) (cdr lis)) (makeset (cdr lis)))  
  ( t (cons (car lis) (makeset (cdr lis)))) ) )  
MAKESET  
> (makeset '(a b a d e b f) )  
(A D E B F)  
> (makeset '(1 1 1 2 2 2 3 3) )  
(1 2 3)
```