

Homework SET #5

Homework 5:Lisp Exercises 3 Originally Due Tuesday September 29, 2009 in class, can be turned in Thursday October 1, 2009.

Tougher LISP Problems. Be sure to follow the guidelines for Programming Assignments. Make sure you give me a short description of the method or approach used, 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. Turn in your hard copy write-up in class but also send me an e-mail copy of your LISP code.

1. An $N \times N$ complex matrix R is represented by N lists of N pairs, where the first element of each pair is the real part of the matrix component and the second element of each pair is the imaginary part of the corresponding matrix component. Give a definition for CADD, a function to add complex matrices using a mapping function.

> (cadd '((1 2) (3 4)) '((5 6) (7 8)))	> (cadd '(((1 2) (3 4)) ((5 6) (7 8))) '(((10 10) (10 10)) ((1 1) (1 1))))
((6 8) (10 12))	((((11 12) (13 14)) ((6 7) (8 9))))
> (cadd '((1 2) (3 4)) ())	> (cadd () '((1 2) (3 4)))
()	nil
> (cadd () ())	> (cadd '((1 2)) '((3 4)))
nil	((4 6))

```
(DEFUN MAPCAR2 (F X Y) (COND
  ((NULL X) NIL)
  ((NULL Y) NIL)
  (T (CONS (FUNCALL F (CAR X) (CAR Y))
    (MAPCAR2 F (CDR X) (CDR Y))))
))
(DEFUN CADD (X Y) (COND
  ((NULL X) NIL)
  ((NULL Y) NIL)
  (T (CONS(MAPCAR2 '+ (CAR X) (CAR Y))
    (MAPCAR2 '+ (CDR X) (CDR Y))))
))
(DEFUN CADD (X Y) (MAPCAR2 '+ X Y))
```

2. Give a definition for CTRANSP, a function to transpose a complex matrix using a mapping function.

> (ctransp '(((1 2) (3 -4)) ((5 6) (7 -8))))	> (ctransp ())
((((1 2) (5 6)) ((3 -4)) ((7 -8))))	Nil
> (ctransp '(((1 2) (3 4)) ((5 6) (7 8))))	> (ctransp '(((10 10) (10 10)) ((1 1) (1 1))))
((((1 2) (5 6)) ((3 4)) ((7 8))))	((((10 10) (1 1)) ((10 10)) ((1 1))))

```
(DEFUN CTRANSP (X)(COND
  ((NULL X) NIL)
  (T (CONS (MAPCAR 'CAR X) (MAPCAR 'CDR X))))
))
```

3. MAPC is like MAPCAR in that it hands to its functional argument (funarg) successive cars of the input list. MAPC differs from MAPCAR in that MAPC prematurely stops applying the funarg to the successive cars of the input list if the value returned by the argument function is NIL. Further, MAPC returns NIL if it was stopped or T if it was allowed to exhaust the input list. This is useful because many of the functions we have defined in LISP should be stopped when the result is obvious and there is no need to traverse the input list until it is exhausted. Give a definition for the MAPC mapping function using a PROG.

```
(defun MAPC (f Lis) (PROG (fvalue)
  LOOP
    ( when (null Lis)          (return T) )
    ( setq  fvalue             (funcall f (car Lis)) )
    ( when (null fvalue)      (return Nil) )
    ( setq  Lis                (cdr Lis) )
    ( go      LOOP ) ) )
```

4. Define ALLNUMS using MAPC. ALLNUMS is a predicate that returns T if its argument is a list of numbers.

```
(defun allnums (lis) (cond
  ((null lis) nil)
  ('else (mapc #'numberp lis))))
```

```
Lisp> (ALLNUMS '(1 3 7)) ==> T
```

5. Define EQSET, a predicate that returns true if two sets are the same set, using the MAPC mapping function.

```
(defun EQSET (s1 s2) (cond
  ((null s1) (null s2))
  ((null s2) nil)
  ( (eq (length s1) (length s2)) (mapc (lambda (el) (member el s2)) s1))))
```

```
Lisp> (eqset '(a b c) '(b a) )           ==>  Nil
Lisp> (eqset '(a b)  '(a b c) )         ==>  Nil
Lisp> (eqset '(a b c) '(b a c) )         ==>  T
Lisp> (eqset '(coke is it) '(is it coke) ) ==>  T
Lisp> (eqset '(inhuman acts are human mistakes) '(human acts are inhuman mistakes) ) ==> T
```

6. Write a lisp program which takes a simple numeric list as input and returns a sorted list in descending order. Do not remove duplications.

```
Lisp> (sort '(3 4 1 0))           ==>      (4 3 1 0)
Lisp> (sort '(1 5 3 2))           ==>      (5 3 2 1)
Lisp> (sort '(3 4 3 2 1 2))       ==>      (4 3 3 2 2 1)
```

This problem may require at least one function using a PROG statement and one or two small, recursive helper functions. You are under no restrictions concerning the sort method (bubble, insertion, radix, quicksort, etc.) you may want to implement.

```
> (defun rotate-left (lis) (append (rest lis) (list (car lis))))
ROTATE-LEFT
> (defun sort (lis) (cond
  ((null lis) nil)
  ((equal (length lis) 1) lis)
  ((mapc #'(lambda (sex) (>= (car lis) sex)) (rest lis)) (cons (car lis) (sort (rest lis))))
  ('else (sort (rotate-left lis)))))
```