MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001 Structure and Interpretation of Computer Programs
Spring. 2007

Practice Problems, March 2

More List Problems Dr. Kimberle Koile

Fill in the code for these recursive procedures.

1. The **append** procedure joins two lists together, e.g., (append (list 1 2) (list 3 4)) => (1 2 3 4)

   a. recursive process

```
(define (append list1 list2)
   (if (null? list1)
       list2
       (cons (car list1)
             (append (cdr list1) list2))))
```

   b. iterative recursive process

```
(define (append list1 list2)
  (define (helper rest result)
      (if (null? rest)
          result
          (helper (cdr rest) (cons (car list1) result))))
   (helper (reverse list1) list 2))
```

2. The **reverse** procedure reverses the order of elements in a list, e.g.,
   (reverse '((1 2) 3 (4 5)) => '((4 5) 3 (1 2)) Hint: Use append.

   a. recursive process

```
(define (reverse x)
   (if (null? x)
       '()
       (append (reverse (cdr x))
               (list (car x)))))
```

   b. iterative recursive process

```
(define (reverse x)
    (define (helper rest result)
        (if (null? rest)
            result
            (helper (cdr rest) (cons (car rest) result))))
     (helper x '()))
```

3. The **deep-reverse** procedure creates a new list with the *every* element in the list recursively reversed as well; e.g., (deep-reverse '((1 2) 3 (4 5)) => '((5 4) 3 (2 1))
(deep-reverse '((1 (2 3)) 4 (5 (6 7)))) => '(((7 6) 5) 4 ((3 2) 1)))

```
(define (deep-reverse lst)
 (cond
   ((null? lst) '())
   ((not (list? (car lst)))
        (append (deep-reverse (cdr lst)) (list (car lst))))
    (else
        (append (deep-reverse (cdr lst)) (list (deep-reverse (car lst)))))))
```