**List Functions**

```
(define (length lst)
  (if (null? lst)
      0
      (+ 1 (length (cdr lst))))
)

(define (map proc lst)
  (if (null? lst)
      '()
      (cons (proc (car lst))
            (map proc (cdr lst))))
)

(define (filter pred lst)
  (if (null? lst)
      '()
      (if (pred (car lst))
          (cons (car lst) (filter pred (cdr lst)))
          (filter pred (cdr lst))))
)

(define (fold-right op init lst)
  (if (null? lst)
      init
      (op (car lst)
           (fold-right op init (cdr lst))))
)

;also known as accumulate, foldr
```

**Problems**

1. Write a function `occurrences` that takes a number and a list and counts the number of times the number appears in the list. Write two versions – one that uses `fold-right`, and one that uses `filter`. For example,

```
(occurrences 1 (list 1 2 1 1 3)) ==> 3

(define (occurrences e lst)
  (length (filter (lambda (x) (= x e)) lst)))
```
(define (occurrences e lst)
 (foldr (lambda (x s)
 (if (= x e)
 (+ s 1)
 s))
 0
 lst)))

2. Define length using a higher order list procedure.

(define (length lst)
 (foldr (lambda (x s)
 (+ s 1))
 0
 lst))

3. Define ls to be a list of *procedures*:

    (define (square x) (* x x))
    (define (double x) (* x 2))
    (define (inc x) (+ x 1))
    (define ls (list square double inc))

Now say we want a function apply-procs that behaves as follows:

(apply-procs ls 4)
=> ((square 4) (double 4) (inc 4)) = (16 8 5)
(apply-procs ls 3)
=> ((square 3) (double 3) (inc 3)) = (9 6 4)

Write a definition for apply-procs using map.

    (define (apply-procs procs value)
     (map (lambda (p) (p value))
          procs))

4. Suppose x is bound to the list (1 2 3 4 5 6 7). Using map, filter, and/or fold-right, write an expression involving x that returns:

(a) (1 4 9 16 25 36 49)
    (map (lambda (e) (* e e)) x)
(b) (1 3 5 7)
    (filter odd? x)
(c) ((1 1) (2 2) (3 3) (4 4) (5 5) (6 6) (7 7))
    (map (lambda (e) (list e e)) x)
(d) \(((2) ((4) ((6) ()')))\)

\[
\text{foldr \ (lambda \ (e \ r) \ (list \ (list \ e) \ r))} \\
\text{\ ()} \\
\text{\ (filter \ even? \ x))}
\]

(e) The maximum element of \(x\): 7

\[
\text{foldr \ max \ 0 \ x}
\]

(f) The last pair of \(x\): (7)

\[
\text{foldr \ (lambda \ (e \ r) \ (if \ (null? \ r) \ (list \ e) \ r))} \ () \ x)
\]