Question 4. (15 points) Write a Scheme function equivalent to the following C function. Your solution does not need to be tail recursive, and it only needs to work for nonnegative integer arguments.

```
int kangaroo(int m, int n)
{
  int val = 3;
  int j;
  for(j=m; j<n; j++){
    val = (val + m) * j;
  }
  return val;
}
```

```
(define (kangaroo m n val j)
   (if (>= j n) val
       (kangaroo m n (* (+ val m) j)
                     (+ j 1))))
```

```
; invoke function as (kangaroo m n 3 m)
; with suitable values for m, n
```

Question 6. (7 points) Curry the Scheme function p below into a Scheme function q. That is, define q such that (p a b) evaluates to the same thing as ((q a) b). You may not use p in your definition of q. You must use only elementary arithmetic functions and standard Scheme keywords.

```
(define (p a b)
  (let ((x (+ b (* a a)))
        (y (- a b)))
   (* x y)))
```

```
(define (q a)
  (lambda (b)
    (let ((x (+ b (* a a)))
          (y (- a b)))
      (* x y))))
```

; invoking the function q : ((q 2) 3) - just an example

Question 8. (8 points) What is the output of the following Scheme code? For each function that produces output, write the output directly to the right of the function that produces the output. The number of points for each output value are listed as comments in the code.

```
(define w (list 3 4 5))
(define x (list 6 7 8))
(define y (list 9 10 11))
(define z (list w x y))

; 1 point
(car x)     6

; 1 point
(cdr x)     (list 7 8)   or   '(7 8)

; 1 point
(car z)     (list 3 4 5)

; 1 point
(cadr z)    (list 6 7 8)

; 1 point
(caadr z)   6

(define (alpha f i j k) (apply f (list i j k)))

; 1 point
(alpha * 2 3 4)   24

(define (beta f i j k) (map f (list i j k)))

; 1 point
(beta add1 2 3 4)   (list 3 4 5)

; 1 point
(apply * (beta add1 2 3 4))   60
```