January 16, 2014
Keng Low

Tail Recursion

```
(define multiply
  (lambda ((a <number>) (b <integer>))
    (cond ((zero? b) 0)
          ((odd? b)
           (+ a (multiply (+ a a) (quotient b 2))))
          (else
           (multiply (+ a a) (quotient b 2))))))
```

Write a version of multiply which uses the same basic algorithm as above, but which is tail-recursive.

```
(define multiply-2
    (lambda ((a <number>) (b <integer>))
      (cond ((zero? b) 0)
            ((odd? b)
             (letrec ((iter
                (lambda ((c <integer>) (result <number>))
                  (if (= c (quotient b 2))
                      result
                      (iter (quotient c 2) (+ result (* (quotient c 2) (+ a a))))))))
                (iter b a)))
            (else
             (letrec ((iter
                (lambda ((c <integer>) (result <number>))
                  (if (= c (quotient b 2))
                      result
                      (iter (quotient c 2) (+ result (* (quotient c 2) (+ a a))))))))
                (iter b 0))))))
```