

```
(define length
  (lambda (l)
    (cond
      ((null? l) 0)
      ((+ 1(length (cdr l)))))))
```

```
(define length
  (lambda (ls)
    (cond
      ((null? ls) 0)
      ((atom? (car ls)) (add1(length (cdr ls))))
      (else(+ (length(car ls)) (length(cdr ls))))
    )
  )
)
```

```
(define length
  (lambda (ls)
    (cond
      ((null? ls) 0)
      (not(atom? (car ls)) (length (car ls)))
      (else(length(cdr ls)))
    )
  )
)
```

```
)  
)
```

```
(define length  
  (lambda (ls)  
    (let ((a 0))  
      (cond  
        ((null? ls) a)  
        (else (+ (length (cdr ls)) 1))))))
```

```
(define atom-count*  
  (lambda (ls)  
    (cond  
      ((null? ls) 0)  
      ((atom? (car ls))(add1(atom-count* (cdr ls))))  
      ((+ (atom-count* (car ls))(atom-count* (cdr ls))))))
```

```
(define atom-count  
  (lambda (l)  
    (cond  
      ((null? l) 0)  
      ((atom?(car l))(add1(atom-count(cdr l))))  
      (else(+1(atom?(car l))(atom-count(cdr l))))))
```

```
(define rdc
  (lambda (ls)
    (reverse (cdr (reverse ls)))))
```

```
(define reverse
  (lambda (ls)
    (let rev ((ls ls) (new '()))
      (if (null? ls) new (rev (cdr ls) (cons (car ls) new))))))
```

```
(define remove-dups
  (lambda (ls)
    (cond
      ((null? ls) '())
      ((null? (cdr ls)) ls)
      ((eq? (car ls) (car (cdr ls)))
       (cons (car ls) (remove-dups (rember (car ls)
                                           (cdr ls)))))
      (else (cons (car ls) (remove-dups (cdr ls))))))
```

```
(define remove-dups
  (lambda (ls)
    (cond
      ((null? ls) '())
      ((null? (cdr ls)) ls)
      (else
       (cond
         ((eq? (car ls) (cadr ls)) (remove-dups (cdr ls))))
       (else (cons (car ls) (remove-dups (cdr ls))))))))
```

```
(define rac
  (lambda (l)
    (cond
      ((eq? (cdr l) '()) (car l))
      (else (rac (cdr l))))))
```