

```
runner% cat length1.1
(defun length1 (list)
  (cond ((null list) 0)
        (t (+ 1 (length1 (cdr list)))))
  )
)
```

```
runner% lisp
.....
> (load "length1.1")
.....
> (length1 ())
0
> (length1 nil)
0
> (length1 '(a b c))
3
> (length1 '((a b) c (d e (f))))
3
```

```
runner% cat myfind.1
(defun myfind (word sent)
  (cond ((null sent) 'not-found)
        ((equal word (car sent)) 'found)
        (t (myfind word (cdr sent)))))
)
```

```
runner% lisp
.....
> (load "myfind.1")
.....
> (myfind 'is '(now is the time))
FOUND
> (myfind 'for '(now is the time))
NOT-FOUND
> (myfind '(a) '(a b c))
NOT-FOUND
> (myfind '(b) '(a (b) c))
FOUND
> (myfind '(c (d e)) '((a b) (c (d e)) e (f)))
FOUND
```

```
runner% cat add.1
(defun add (list)
  (cond ((null list) 0)
        (t (+ (car list) (add (cdr list)))))
  )
)
```

```
> (add '(2))
2
> (add '(2 3))
5
> (add '(2 3 4))
9
> (add nil)
0
> (add 3)
>>Error: The value of X, 3, should be a LIST
Back to Lisp Top Level
> (add '(2 (3 4)))
>>Error: The value of NUMBER1, (3 4), should be a NUMBER
```

```
runner% cat add-all.1
(defun add-all (list)
  (cond ((null list) 0)
        ((atom (car list)) (+ (car list) (add-all (cdr list))))
        (t (+ (add-all (car list)) (add-all (cdr list)))))
  )
)
```

```
> (add-all '(1))
1
> (add-all '(2 3))
5
```

## CS 3723, Simple examples of recursion in Lisp, Page 2

```
> (add-all '(3 4 5))
12
> (add-all '(2 (3 4)))
9
> (add-all '(( (2 3) 4) (1 5)))
15
```

---

```
runner% cat list-atoms.l
(defun list-atoms (list)
  (cond ((null list) nil)
        ((atom (car list)) (cons (car list) (list-atoms (cdr list))))
        (t (append (list-atoms (car list)) (list-atoms (cdr list))))))
> (list-atoms ())
NIL
> (list-atoms '(a))
(A)
> (list-atoms '(a b))
(A B)
> (list-atoms '(a (b c)))
(A B C)
> (list-atoms '((a b) c (d (e (g) f))))
(A B C D E G F)
> (list-atoms '(() a))
(NIL A)                                     *** NOT CORRECT ANSWER ***
> (list-atoms (car '(() a)))
NIL
> (list-atoms (cdr '(() a)))
(A)
> (append nil '(a))
(A)
> (atom (car '(() a)))
T> (atom ())
T                                             *** OK, HERE'S THE PROBLEM ***
> (cons (car '(() a)) (list-atoms (cdr '(() a))))
(NIL A)
```

---

```
runner% cat list-atoms.l
(defun list-atoms (list)
  (cond ((null list) nil)
        ((null (car list)) (list-atoms (cdr list)))
        ((atom (car list)) (cons (car list) (list-atoms (cdr list))))
        (t (append (list-atoms (car list)) (list-atoms (cdr list))))))
> (list-atoms ())
NIL
> (list-atoms '(a))
(A)
> (list-atoms '(a (b c)))
(A B C)
> (list-atoms '((a b) c (d (e (g) f))))
(A B C D E G F)
> (list-atoms '(() a b))
(A B)
```

---

```
runner% cat list-atoms.l
(defun list-atoms (list)
  (cond ((null list) nil)
        ((and (not (null (car list))) (atom (car list)))
         (cons (car list) (list-atoms (cdr list))))
        (t (append (list-atoms (car list)) (list-atoms (cdr list))))))
> ; note: how append and cons handle nil
(append nil '(a) nil)
(A)
> (cons 'a nil)
(A)
```