

# Computer Science I

Professor Tom Ellman  
Lecture 7

## Length of a List

```
(define length (lambda (lst) ...?...))
```

Welcome to DrScheme.

```
> (length '(a b c))
```

```
3
```

```
> (length '(a (b c)))
```

```
2
```

```
> (length '())
```

```
0
```

## Length of a List

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

## Procedure Call Tree

(length '(a (b c)))

(length '((b c)))

(length '())

Parent node value results  
from incrementing child  
node value.

2

1

0

(length '(a (b c)))

(length '((b c)))

(length '())

## Counting the Items in an Expression

```
(define item-count (lambda (x) ...?...))
```

Welcome to DrScheme.

```
> (item-count '(a b c))
```

```
3
```

```
> (item-count '(a (b c)))
```

```
3
```

```
> (item-count 'a)
```

```
1
```

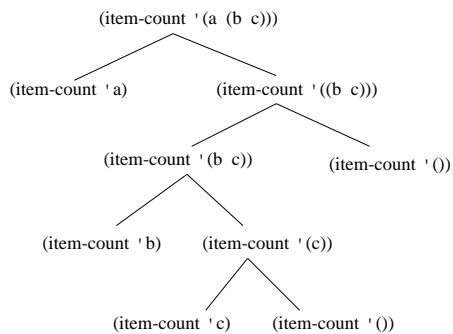
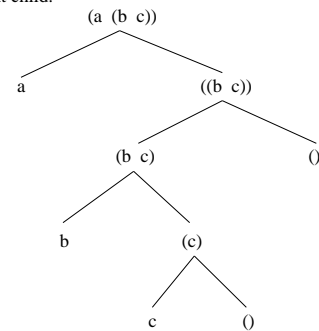
```
> (item-count '())
```

```
0
```

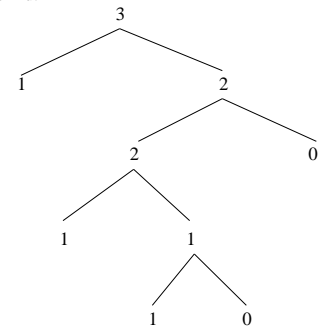
## Counting the Items in an Expression

```
(define item-count
  (lambda (x)
    (cond ((null? x) 0)
          ((not (pair? x)) 1)
          (else (+ (item-count (car x))
                   (item-count (cdr x)))))))
```

Parent is the result of CONSing  
the left child into the right child.



Parent is the result of adding  
the left child to the right child.



## Data Driven Recursion

- The data has the structure of a tree.
- The calls to a recursive procedure have the structure of a tree.
- The two trees have the *same* structure.

## Flat Recursion

- A call to (procedure lst) may lead to a recursive call to (procedure (cdr lst)).
- The calls form a linear tree.
- Each tree node has zero or one child nodes.
- Sometimes called “cdr recursion”.
- Example: The LENGTH procedure.

## Deep Recursion

- A call to (procedure lst) may lead to a recursive call to (procedure (car lst)) and (procedure (cdr lst)).
- The calls form a binary tree.
- Each tree node has zero or two child nodes.
- Sometimes called “car cdr recursion”.
- Example: The ITEM-COUNT procedure.

## Is an Item a Member of a List?

```
(define member? (lambda (item lst) ...?...))
```

Welcome to DrScheme.

```
> (member? 'b '(a b (c) d))
```

```
#t
```

```
> (member? 'c '(a b (c) d))
```

```
#f
```

```
> (member? 'groucho '())
```

```
#f
```

## Is an Item a Member of a List?

```
(define member?  
  (lambda (item lst)  
    (and (not (null? lst))  
         (or (equal? item (car lst))  
             (member? item (cdr lst))))))
```

## Is an Item Found Anywhere Within an Expression?

```
(define within? (lambda (item x) ...?...))
```

Welcome to DrScheme.

```
> (within? 'b '(a b (c) d))
```

```
#t
```

```
> (within? 'c '(a b (c) d))
```

```
#t
```

```
> (within? 'c 'c)
```

```
#t
```

```
> (within? 'groucho '())
```

```
#f
```

## Is an Item Found Anywhere Within an Expression?

```
(define within?  
  (lambda (item x)  
    (or (equal? item x)  
        (and (pair? x)  
             (or (within? item (car x))  
                 (within? item (cdr x))))))
```

## Appending two Lists

```
(define append (lambda (lst1 lst2) ...?...))
```

Welcome to DrScheme.

```
> (append '(a b c) '(d e f))
```

```
(a b c d e f)
```

```
> (append '() '(a b c))
```

```
(a b c)
```

```
> (append '(a b c) '())
```

```
(a b c)
```

## Appending two Lists

```
(define append
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (cons (car lst1) (append (cdr lst1) lst2)))))
```

## Flatten an Expression

```
(define flatten (lambda (x) ...?...))
```

Welcome to DrScheme.

```
> (flatten '(a (b) c))
```

```
(a b c)
```

```
> (flatten '(a b c))
```

```
(a b c)
```

```
> (flatten 'z)
```

```
(z)
```

```
> (flatten '())
```

```
()
```

## Flatten an Expression

```
(define flatten
  (lambda (x)
    (cond ((null? x) '())
          ((not (pair? x)) (list x))
          (else (append (flatten (car x))
                        (flatten (cdr x)))))))
```

## The Depth of an Expression

```
(define depth (lambda (x) ...?...))
```

Welcome to DrScheme.

```
> (depth 'b)
```

```
0
```

```
> (depth '(a b c))
```

```
1
```

```
> (depth '(a (b) c))
```

```
2
```

## The Depth of an Expression

```
(define depth
  (lambda (x)
    (cond ((null? x) 1)
          ((not (pair? x)) 0)
          (else (max (+ 1 (depth (car x)))
                    (depth (cdr x)))))))
```