Flat Recursion

- A call to (function lst) may lead to a recursive call to (function (rest lst)).
- The calls form a linear tree.
- Each tree node has zero or one child nodes.
- Example: The LENGTH function.

Length of a Flat List

(define (length lst) ...

Welcome to DrRacket.
> (length '(a  b  c))
3
> (length '(a (b  c)))
2
> (length '())
0

Length of a Flat List

(define (length lst)
  (if (empty? lst) ;; base case
      lst
      empty
    0
    ;; recursive case, add 1 for first
    ;; item and call length on rest
    (+ 1 (length (rest lst)))))

Length of a Flat List

(define (length lst)
  (if (empty? lst) ;; base case
      lst
      empty
    0
    ;; recursive case, add 1 for first
    ;; item and call length on rest
    (+ 1 (length (rest lst)))))

Procedure Call Tree

(length '(a (b c)))
  (length '((b c)))
  (length '())

Deep Recursion

- A call to (function lst) may lead to a recursive call to (function (first lst)) and (function(rest lst)).
- The calls form a binary tree.
- Each tree node has zero or two child nodes.
- Example: The ITEM-COUNT procedure.

Parent node value
results from
incrementing child
node value.

<table>
<thead>
<tr>
<th>Node Value</th>
<th>(length '(a (b c)))</th>
<th>(length '((b c)))</th>
<th>(length '())</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Counting the Items in an expression

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

Welcome to DrScheme.

```scheme
> (item-count '(a b c))
3
> (item-count '((a (b c))))
3
> (item-count 'a)
1
> (item-count '())
0
```

```scheme
(define (item-count x)
    ; base case 1: 0 items
    (cond [(null? x) 0]
          ; base case 2: not a list, return 1
          [(not (cons? x)) 1]
          ; recursive case: sum the result
          ; of calling on first and rest
          [else (+ (item-count (first x))
                   (item-count (rest x))))])
```

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

Recursion over deep lists

- 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.
Is an Item a in a Flat List?

(define (member? 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 Flat List?

(define (member? item lst)
  (cond
   [(empty? lst) #f]
   [(equal? item (first lst)) #t]
   [else (member? item (rest lst))]))

Flatten an expression into a flat list

(define (flatten x) ...?)

Welcome to DrScheme.
> (flatten '(a (b) c))
(a b c)
> (flatten '(a b c))
(a b c)
> (flatten '(x))
(x)
> (flatten '())
()