CS 135 Winter 2018
Tutorial 7: Accumulative Recursion and Binary Trees
Goals of this tutorial

You should be able to...

• understand the difference between structural and accumulative recursion.

• write functions using accumulative recursion.

• understand the difference between binary trees and binary search trees.

• perform structural recursion on binary trees and binary search trees.
Review: Structural vs. Accumulative

In (pure) structural recursion, all arguments to the recursive call(s) are either unchanged, or *one step* closer to a base case.

In accumulative recursion, arguments are the same as above, plus one or more accumulators, or arguments containing partial answers. The accumulatively recursive function is a helper function, and a wrapper function sets the initial value of the accumulator(s).

If a parameter is used to produce the answer in the base case, then that parameter is probably an *accumulator*. 
**CQ:** Consider the following function:

```scheme
(define (mystery m n lst)
  (cond
    [(empty? lst) (list 1412)]
    [(<= m -72) (list 7)]
    [(>= n 48) (list 3)]
    [else (cons (* m n (first lst))
                 (mystery (sub1 m) (add1 n) (rest lst)))]))
```

The type of recursion of this function is:

A pure structural recursion  
B accumulative recursion  
C generative recursion
Group Problem: separate-sums-within?

Write a function `separate-sums-within?` that consumes two numbers: `split-point` and `threshold`, and a list of numbers `lon`. The function produces `true` if the difference between (the sum of all the numbers in `lon` less than `split-point`) and (the sum of all the numbers in `lon` greater than `split-point`) is within the tolerance of `threshold`, and `false` otherwise. Your solution must use at most one helper function and **accumulative recursion**.

\[
\text{(separate-sums-within? 9 2 (list 10 7 9 8 2 6)) } \Rightarrow \text{ true}
\]
\[
\text{(separate-sums-within? 5 3 (list 2 4 6 8)) } \Rightarrow \text{ false}
\]
Group Problem: remove-leaf/bt

(define-struct node (key val left right))
;; A Node is a (make-node Num Str BT BT)

;; A Binary Tree (BT) is one of:
;; ★ empty
;; ★ Node

Write a function remove-leaf/bt that consumes a number and a binary tree. The consumed number must only be a key for leaf nodes – it cannot be a key for any internal nodes. The function removes all leaf nodes in the tree with the same key as the consumed number. If none of the leaf nodes in the consumed tree have the same key as the consumed number, the same tree is produced.
Review: BST Data Definition

Recall the data definition of a Binary Search Tree (BST):

```scheme
(define-struct node (key val left right))
;; A Node is a (make-node Num Str BST BST)
;; requires: key > every key in left BST
;; key < every key in right BST

;; A Binary Search Tree (BST) is one of:
;; ★ empty
;; ★ Node
```
Clicker Question: Create BST

Which BST would be created from the list \(\{11, 29, 15, 13, 9, 10\}\) by inserting all numbers in this list into an empty BST in reverse order (which means the last element in this list would be first inserted)?

A  A
B  B
C  C
D  None of the above
Clicker Question: BST Insertion

Where would you add 4 to the above BST?

A  A
B  B
C  C
D  Any of the above
Group Problem: remove-leaf/bst

(define-struct node (key val left right))

;; A Node is a (make-node Num Str BST BST)

;; requires: key > every key in left BST

;; key < every key in right BST

;; A Binary Search Tree (BST) is one of:

;; ★ empty

;; ★ Node

Write a function remove-leaf/bst that consumes a number and a binary search tree. The function will do the same thing as the remove-leaf/bt function, with the same restrictions on the leaf nodes. Do you need to traverse the entire tree again?
Group Problem: path-to-key

Write a function `path-to-key` that consumes a number and a binary search tree, and produces a list of the keys along the path from the root node to the node with the same key as the consumed number. If none of the nodes in the consumed tree have the same key as the consumed number, the function should produce `false`. You may find it helpful to use the `search-bst` function on slide 29 of module 8.

```scheme
(define sample-bt (make-node 4 "a" (make-node 2 "b" empty empty)
              (make-node 9 "c"
              (make-node 6 "d" empty empty)
              empty)))
```

```scheme
(path-to-key 6 sample-bt) ⇒ (list 4 9 6)
(path-to-key 8 sample-bt) ⇒ false
```
(define (search-bst n t)
  (cond [(empty? t) false]
        [(= n (node-key t)) (node-val t)]
        [(< n (node-key t)) (search-bst n (node-left t))]
        [(> n (node-key t)) (search-bst n (node-right t))])))