CS 135 Winter 2017

Tutorial 7: Trees
CS 135 Data Progression

Here is a visual representation of the kind of data studied in CS 135.

Each element:
- value
- child1 (rest)

We've studied these

Now we're studying these

CS135 Winter 2017  Tutorial 7: Trees
One can think of a value as a type of tree where each element has 0 children. For a simple manipulation of such a value, no recursion is necessary.

A list can be thought of as a tree where each element has only 1 child, and so the template has 1 recursive call for a function that consumes a list.

Following this pattern, for a binary tree, each element (or node) has 2 children, so the template contains 2 recursive calls.

And finally, for an unbounded tree, each element (or node) has a list of children, so the template has $x$ recursive calls: one for each child. To achieve this, another function must help go through that list of children.
Clicker Question - Create BST

Which BST would be created from the list ’(17 16 13 14 12 5 11 15) 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 - bst-max-min

Write a function bst-max-min which consumes a BST and (any of 'max 'min) and produces the max or min key of the tree, depending on the given symbol. If the tree is empty, produce false. The purpose, contract and examples are given below.
Group Problem - is-bst?

Write a predicate is-bst? which consumes Any and produces true if the argument is a valid BST, otherwise produces false. Hint: Use bst-max-min as a helper function. The purpose and contract are given below.

;; (is-bst? tree) determines whether tree is a valid binary search tree
;; is-bst?: Any → Bool
Group Problem - Languages

```
(define-struct lang (name descend))
;; A Lang is a (make-lang Str (listof Lang))
```

Given the definition of a structure `Lang` representing information about a language, draw the following hierarchy of languages as a tree:

```
(make-lang "Indo-European"
  (list (make-lang "Germanic" (list (make-lang "English" empty)))
    (make-lang "Italic"
      (list (make-lang "Latin"
        (list (make-lang "French" empty)
          (make-lang "Spanish" empty)
          (make-lang "Italian" empty))))))

(make-lang "Slavic"
  (list (make-lang "Polish" empty)
    (make-lang "Russian" empty)))
```
Group Problem - Languages

Indo-European

Germanic
  English

Italic
  Latin
  French  Spanish  Italian

Slavic
  Polish  Russian
Review: AINode structure

Recall the structure and data definition of AINode

```
(define-struct ainode (op args))
;; An Arithmetic expression Internal Node (AINode)
;; is a (make-ainode (anyof ’* ’+) (listof AExp))

;; An Arithmetic Expression (AExp) is one of:
;; ★ a Num
;; ★ an AINode
```
\[
\text{(make-ainode ' + (list (make-ainode ' * (list 4 2)))
3
\text{(make-ainode ' + (list 5 1 2)))
2))}
\]
Group Problem - count-addition

Write a function `count-addition` which consumes an `AExp` and produces the number of times the addition function is applied throughout the tree. Include a contract for `count-addition` and for any other helper functions you write.

;; Examples:
(define ainode1 (make-ainode '⋆ (list 4 2)))
(define ainode2 (make-ainode '+ (list 5 1 2)))

(check-expect (count-addition (make-ainode '+ (list ainode1 3 ainode2)))
  2)