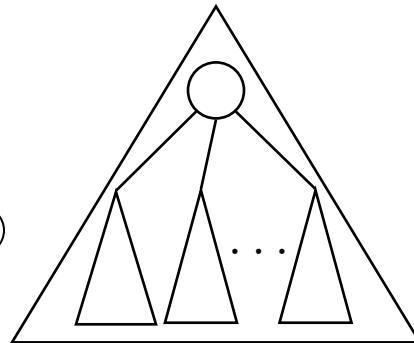
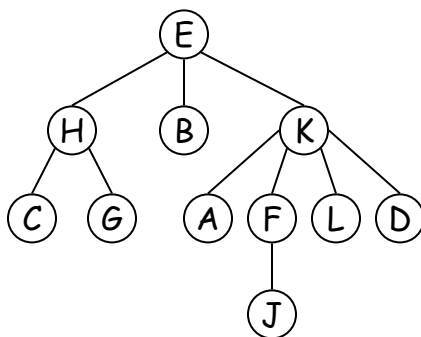


A New Math Type: Tree



Math Tree Continued...



Math Definition for Tree of A

- Base case:

If x is an element of A , then *compose* (x , *empty_string*) is an element of *tree of A*.

- Inductive case:

If T_1, T_2, \dots, T_k are elements of *tree of A* and if x is an element of A , then *compose* ($x, \langle T_1, T_2, \dots, T_k \rangle$) is an element of *tree of A*.

Tree vs. Binary Tree

- Obvious: binary trees have max 2 children restriction, trees don't
- Less obvious: there is no empty tree (as opposed to empty binary tree)
 - Smallest tree has size 1 and no children

Math Operations

- Let $T = \text{compose}(x, \langle T_1, T_2, \dots, T_k \rangle)$
 - $\text{size}(T) \equiv |T| = |T_1| + |T_2| + \dots + |T_k| + 1$
 - $\text{root}(T) = x$
 - $\text{children}(T) = \langle T_1, T_2, \dots, T_k \rangle$
- Let $s = \langle x_1, x_2, \dots, x_k \rangle$
 - $\text{first}(s) = x_1$
 - $\text{last}(s) = x_k$

Tree Component

- Type
 - **Tree_Kernel** is modeled by
tree of Item
- Initial Value
 - **there exists** x : Item
($\text{is_initial}(x)$ and
 $\text{self} = \text{compose}(x, \text{empty_string})$)



Tree Continued...

- Operations
 - t.Add (pos, subtree)
 - t.Remove (pos, subtree)
 - t.Number_Of_Children ()
 - t.Size ()
 - t[current] (accessor)



Practice Operation

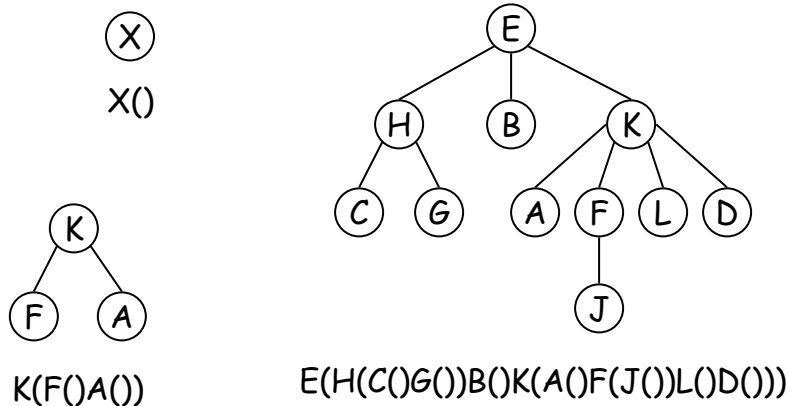
- Most operations on Tree have to be recursive
- Use 5 step process to recursion:
 0. State the problem
 1. Visualize recursive structure
 2. Verify that visualized recursive structure can be leveraged into an implementation
 3. Visualize a recursive implementation
 4. Write a skeleton for the operation body
 5. Refine the skeleton into an operation body

Step 0:

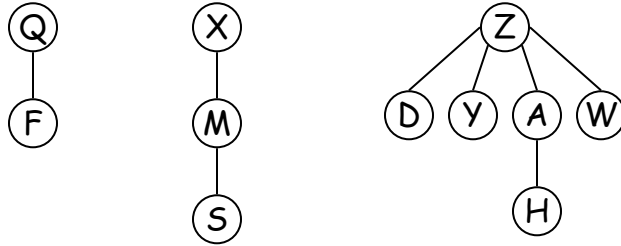
State the Problem

```
procedure Display_Tree (  
  preserves Tree_Of_Integer& t,  
  alters Character_OStream& outs  
);  
/*!  
  requires outs.is_open = true  
  ensures outs.is_open = true and  
    outs.ext_name = #outs.ext_name and  
    outs.contents =  
      #outs.contents * OUTPUT_REP (t)  
!*/
```

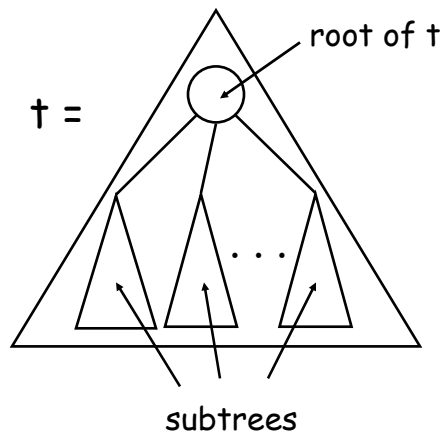
What's OUTPUT_REP (t)?



You Give It a Try!



Step 1: Visualize Recursive Structure



Step 2:

Verify That Leveraging Works

- Ask yourself: If `Display_Tree` could get a helper to display the subtrees, could it take advantage of this generous offer?
- Yes! Once you know how to display the subtrees, you can just display the root followed by the subtrees between '(' and ').

Step 3:

Visualize Recursive Process

Processing non-smallest incoming values of `t`:

Processing smallest incoming values of `t`:

