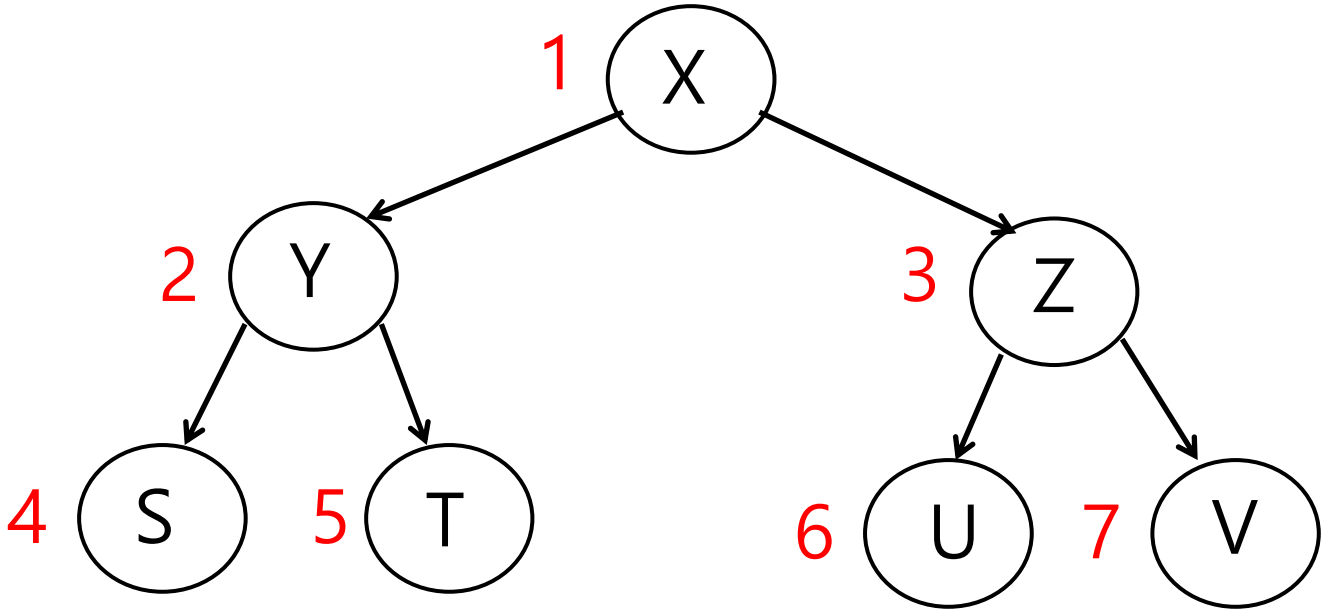
Data Structures:
Trees: Binary Trees, Binary Expression Trees

YoungWoon Cha
(Slide credits to Won Kim)
Spring 2022

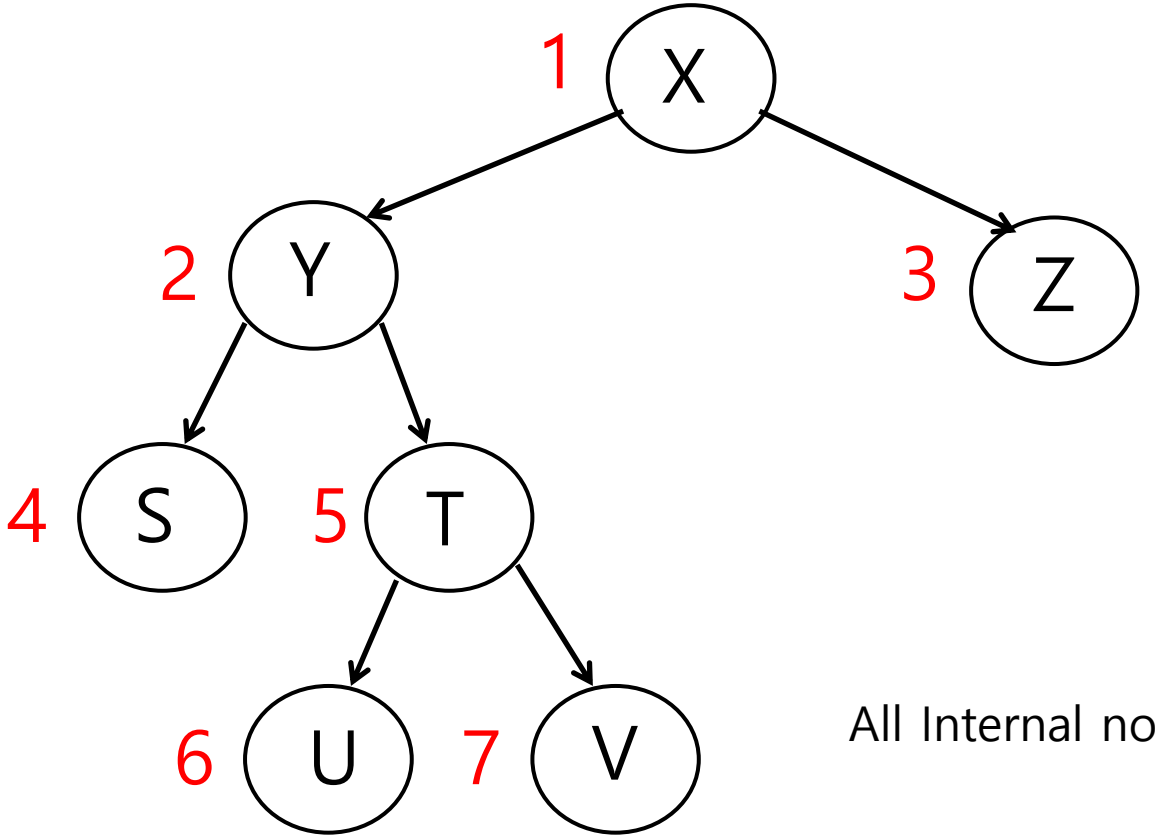
Binary Trees

A Perfect Binary Tree



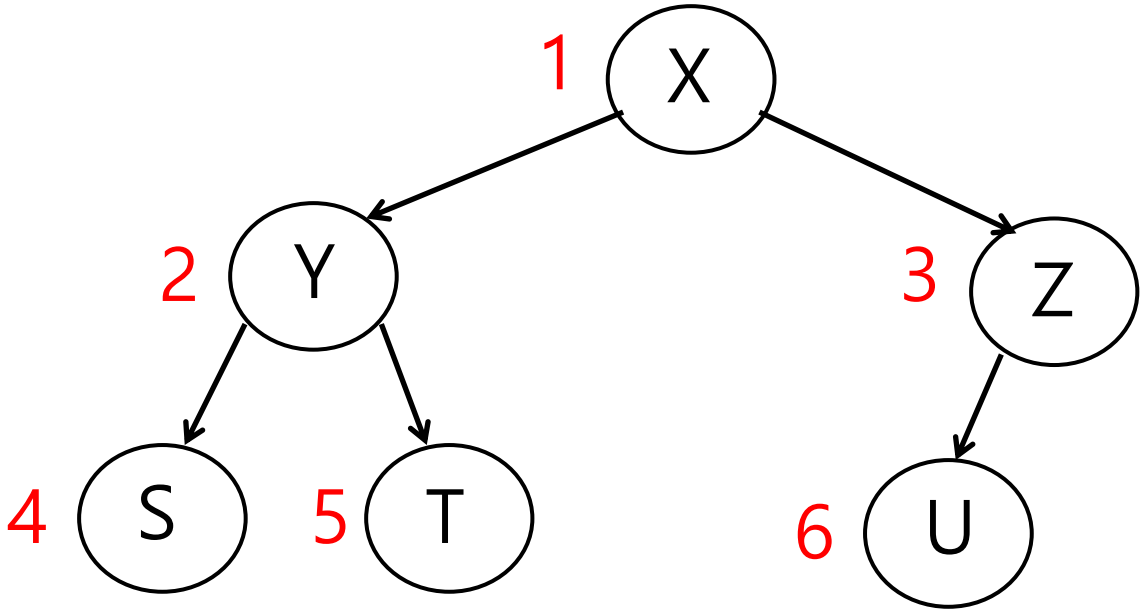
All Internal nodes have degree = 2,
All leaf nodes are at the same level.

A Full Binary Tree



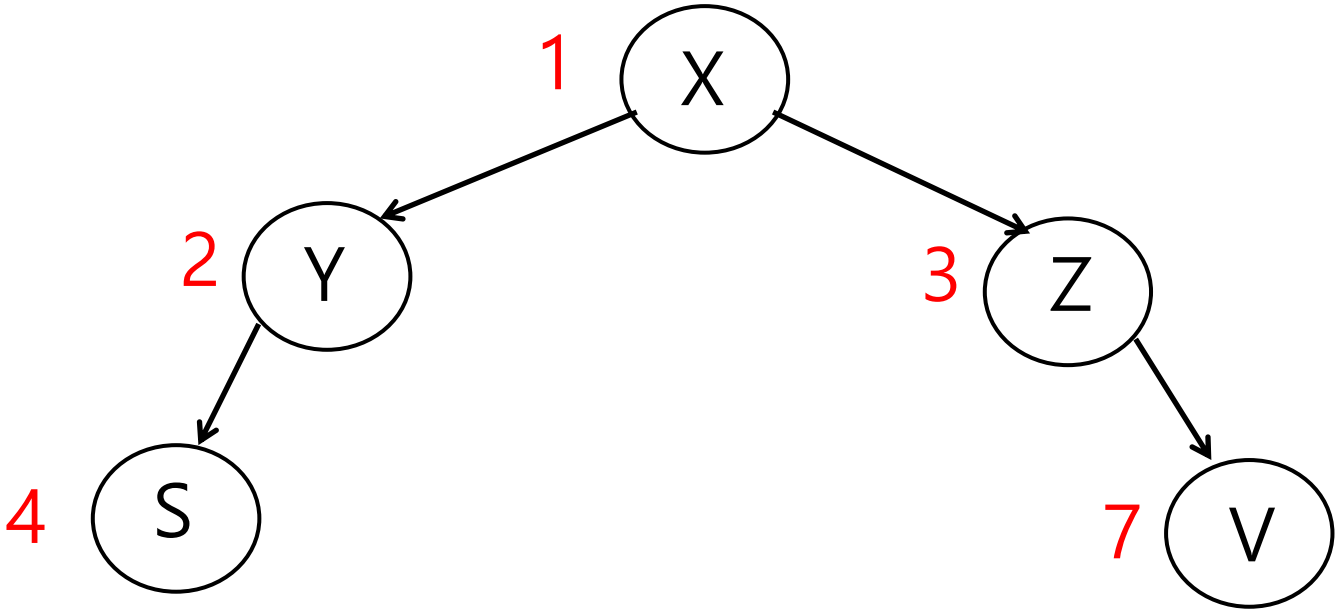
All Internal nodes have degree = 2

A Complete Binary Tree



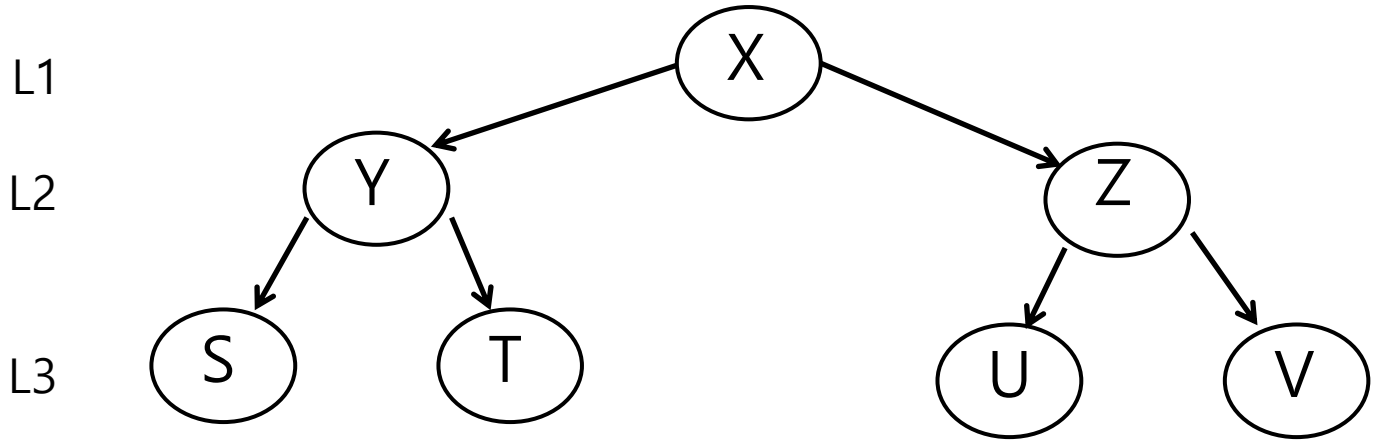
All the levels except the lowest one (1~N-1 levels) are completely filled.
The lowest level (Level N) is filled from the left.

Not a Complete Binary Tree
(* middle teeth missing ^ ^ *)

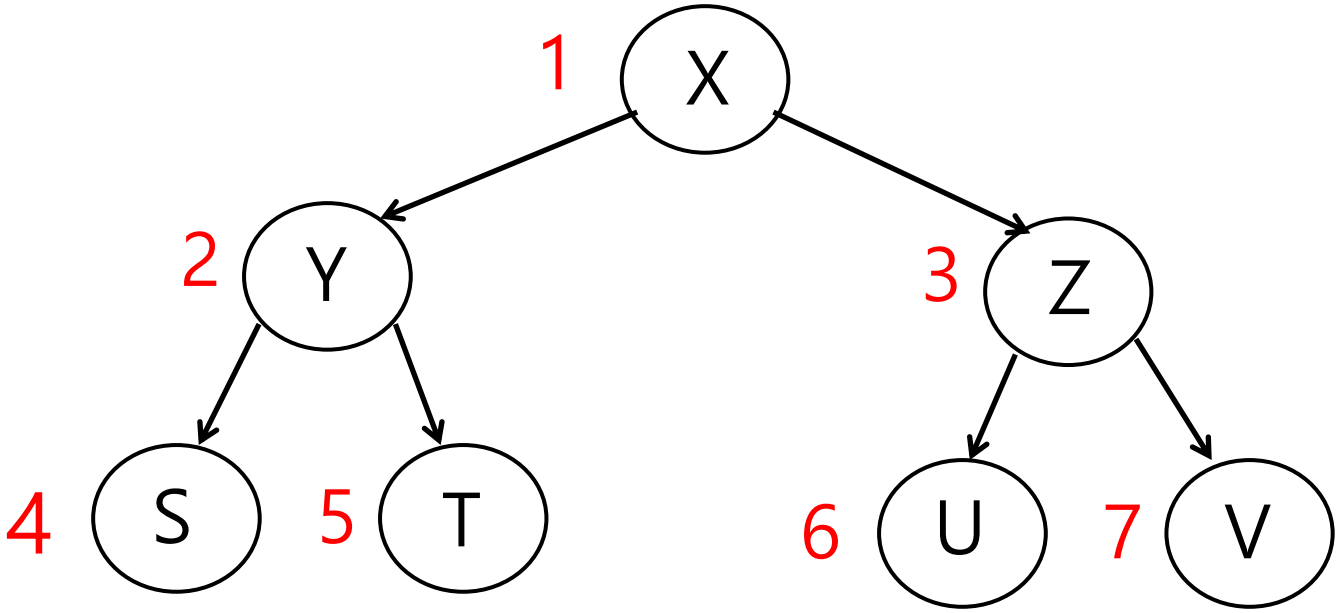


Binary Tree: Properties

- The degree of each non-leaf node is one or two.
- The maximum number of nodes on the i-th level of a tree = 2^{i-1} (i=3, max num= 4)
- The maximum total number of nodes in a tree of height h = $2^h - 1$ (h=3, max total=7)



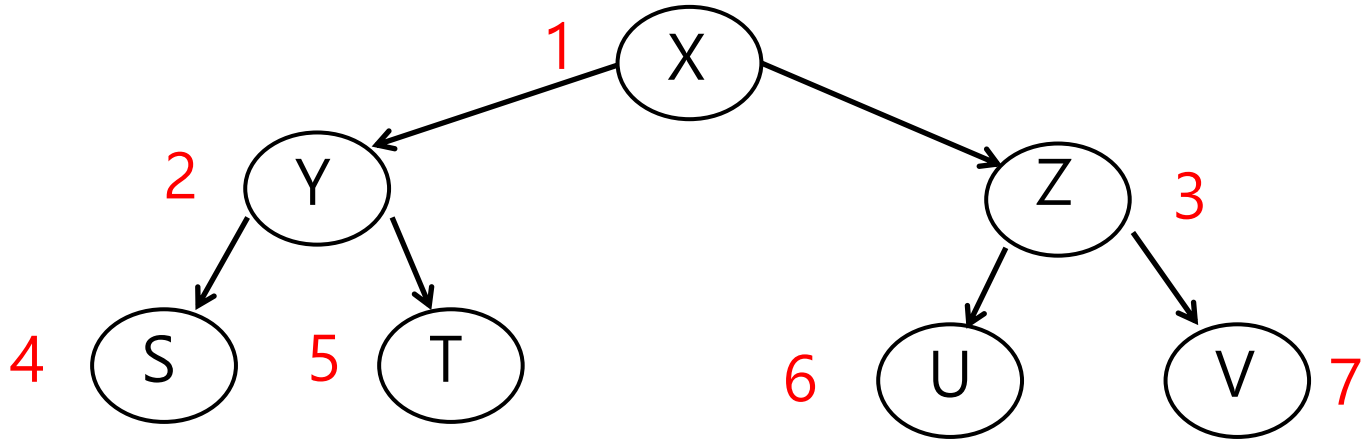
Implementing a Binary Tree Using an Array



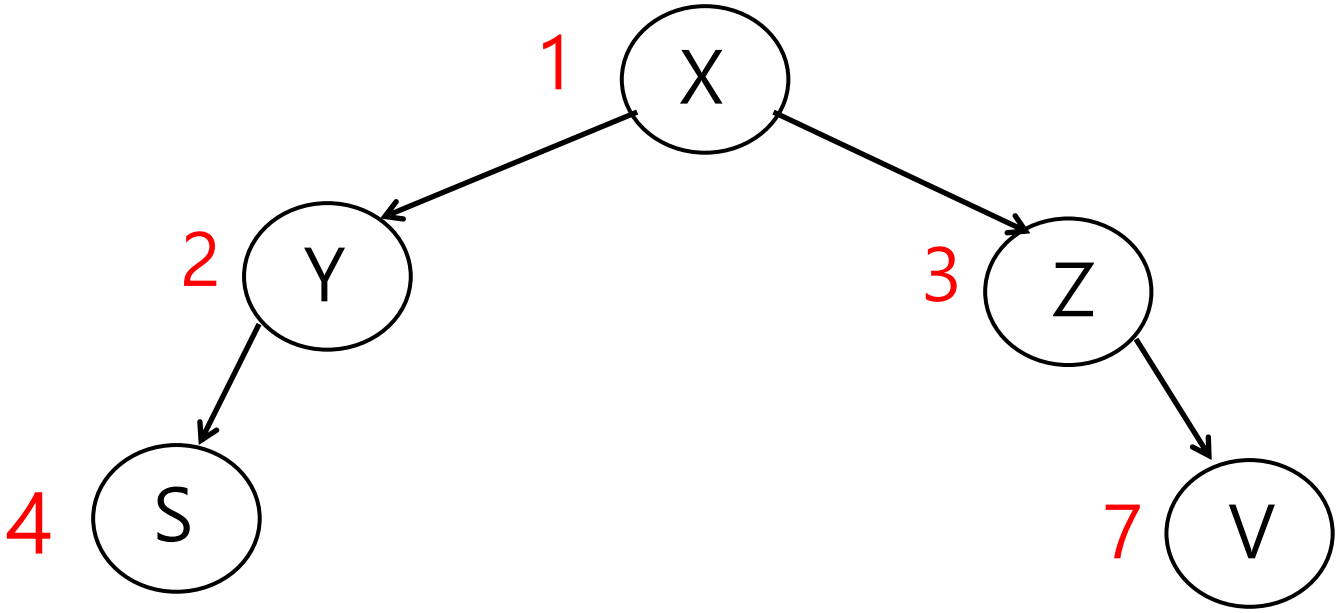
1	X
2	Y
3	Z
4	S
5	T
6	U
7	V

Computing the Locations of Nodes

- For a node with array index i on a full binary tree with n nodes
  - parent of i = ⌊i/2⌋
  - left child of i = 2i
  - right child of i = 2i + 1



Implementing a Binary Tree Using an Array

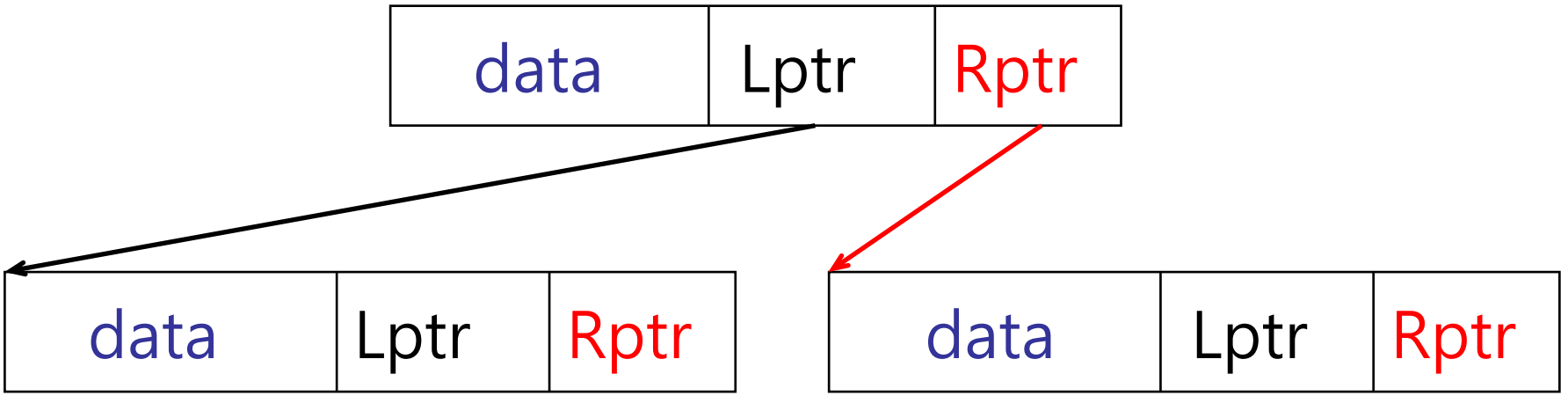


1	X
2	Y
3	Z
4	S
5	
6	
7	V

Problems

- Insertion or deletion of nodes in the middle of an array is expensive.
- Memory is wasted.
  - When the tree is not full/complete

Implementation of a Binary Tree Using a Singly Linked List With 2 Pointers



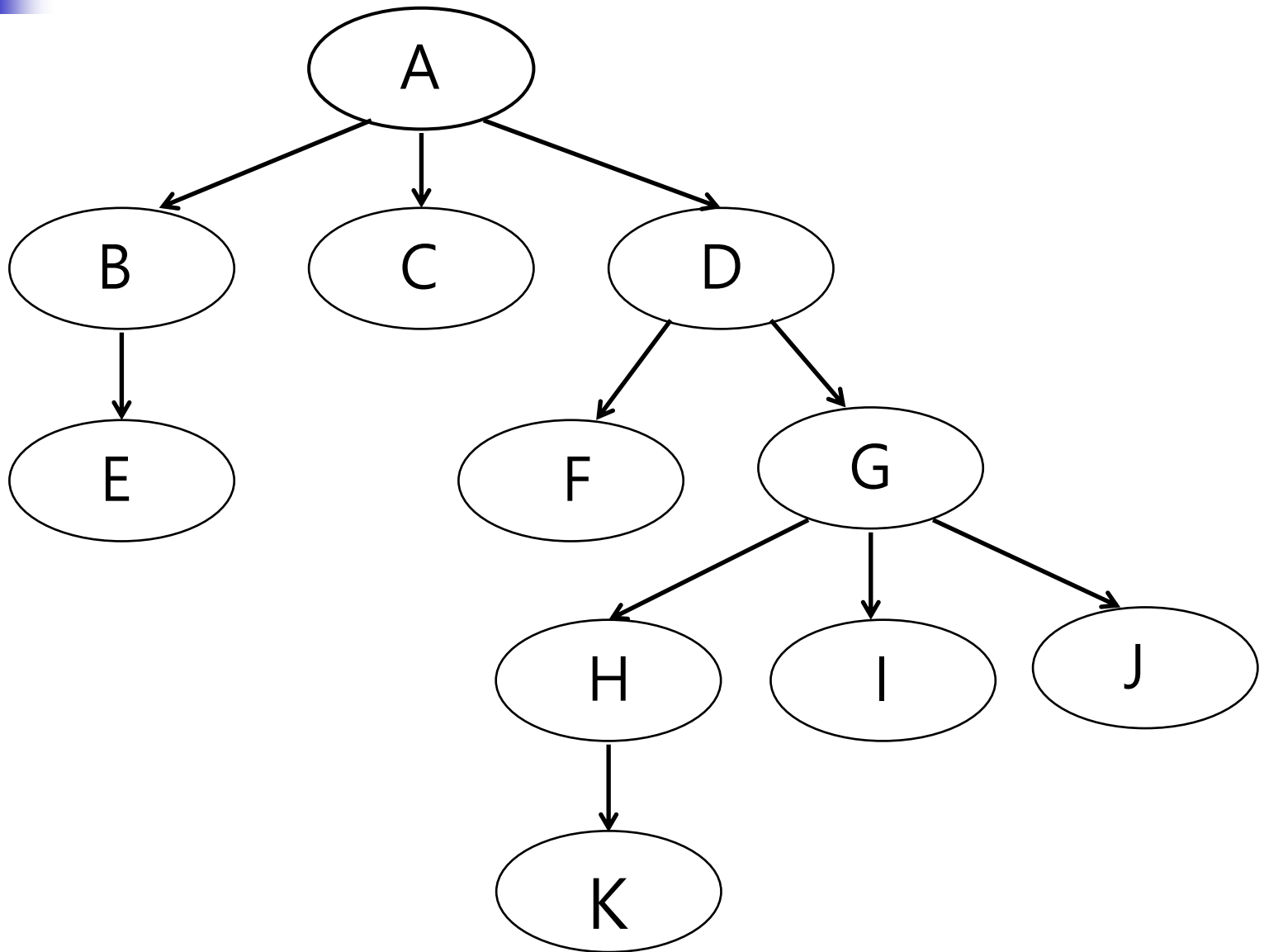
Coding in C

- **Each Node as a Structure**
  - **(data, left-child ptr, right-child ptr)**

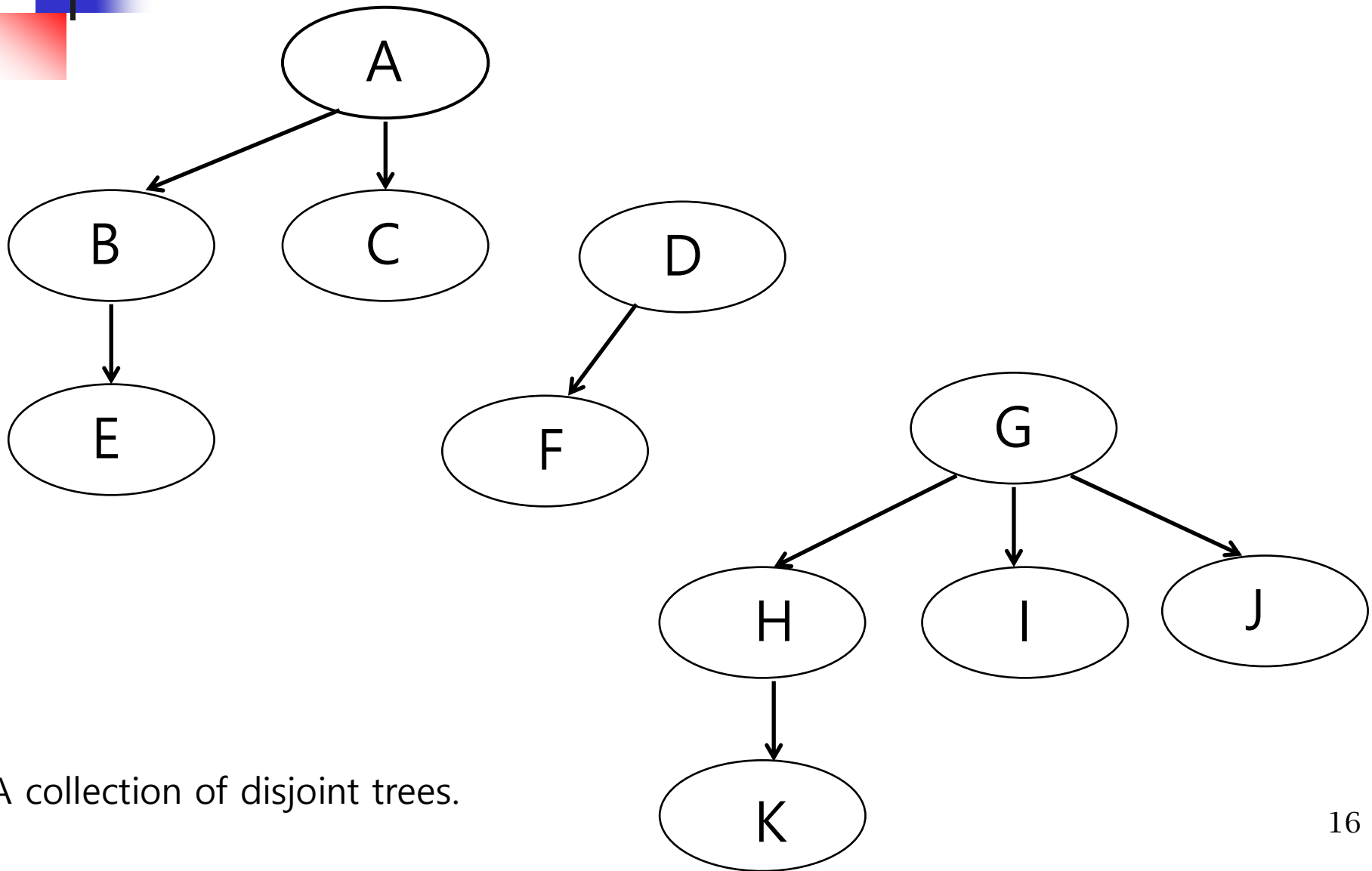
```
typedef  struct  node *tree_ptr;
typedef  struct  node  {
                 (data_type    data);
                 tree_ptr   left_child, right_child;
                 };
```

Representations of Trees

General Tree (Rooted Tree)

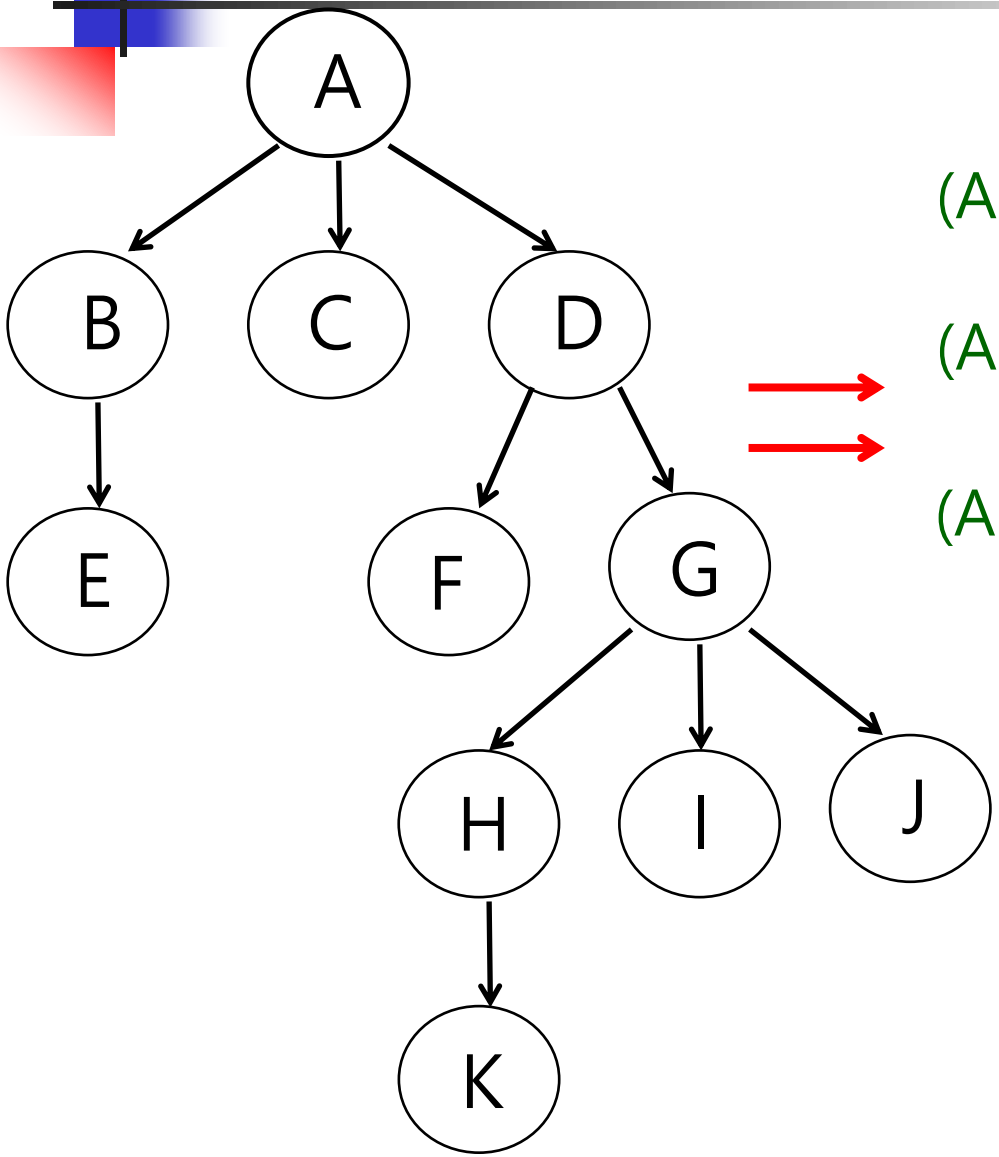


Forest (of 3 Trees)



A collection of disjoint trees.

List Representation of a Tree



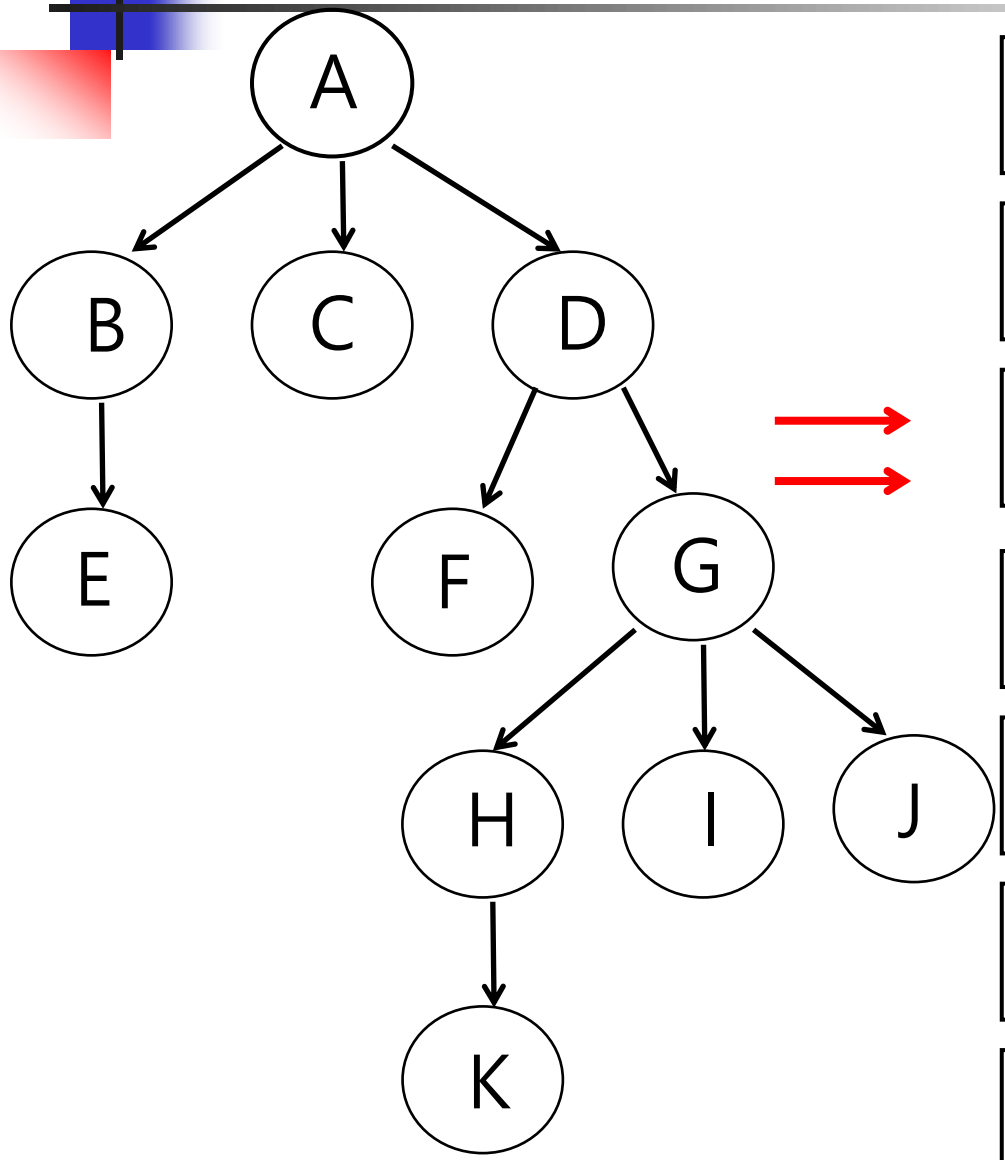
(A(B,C,D))

(A(B(E),C,D(F,G)))

(A(B(E),C,D(F,G(H,I,J))))

(A(B(E),C,D(F,
G(H(K),I,J))))

Linked List Representation of a Tree (In Memory) (* pB, etc. is pointer to B, etc.)



A	pB	pC	pD	

B	pE	

C	

D	pF	pG	

E	

F	

G	pH	pI	pJ	

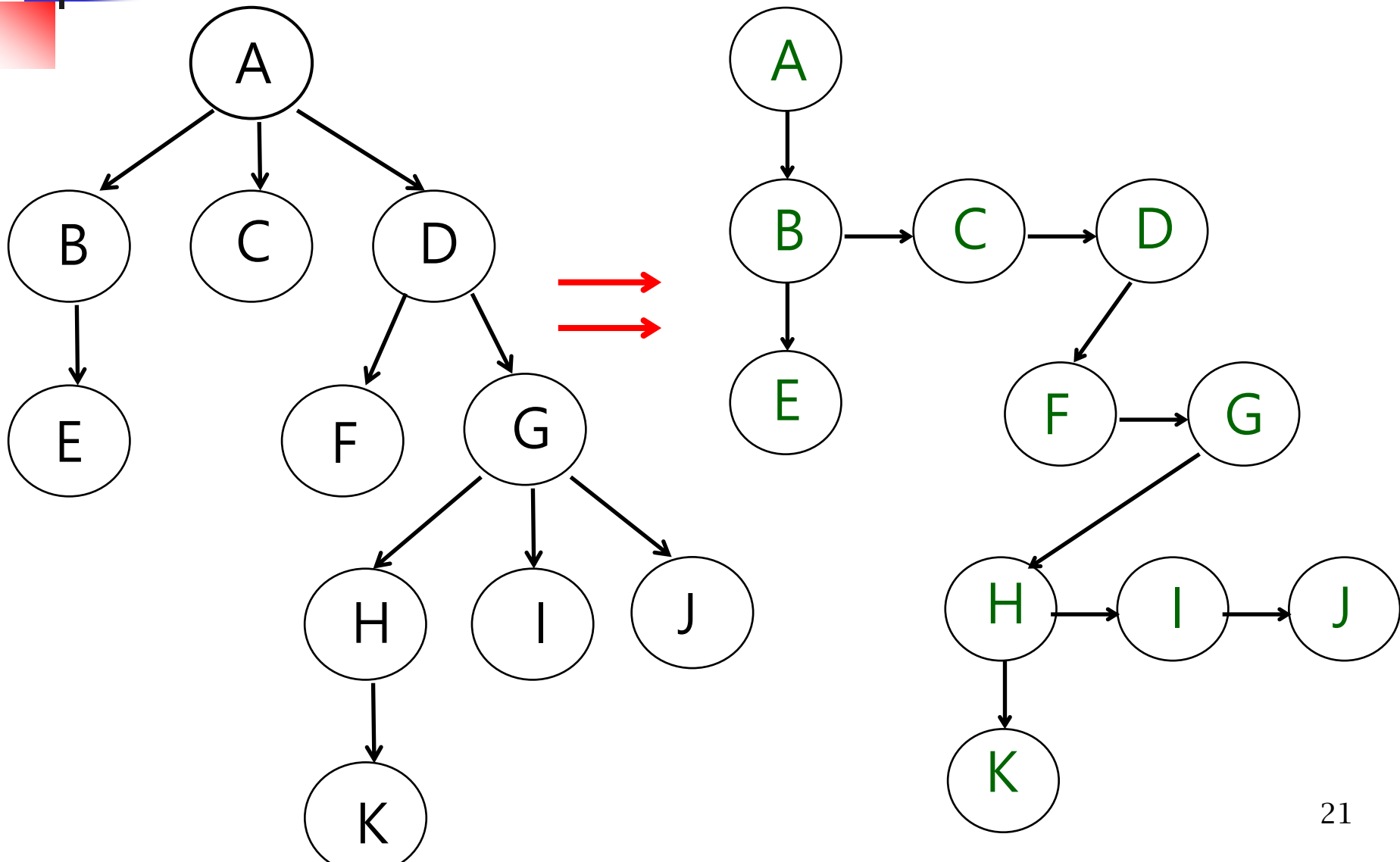
Converting a General Tree into a Binary Tree

- The varying degree of the node in a general tree makes it difficult to read, insert, and delete nodes.
- It is best to transform a general tree into a binary tree.
- We can use the "Leftmost Child-Right Siblings" Representation

Leftmost Child-Right Siblings Representation

- Step 1
  - Connect all siblings of a parent, and delete all links from the parent to its children (except for the one to the leftmost child).
- Step 2
  - Make the right sibling the right child.

Illustration of Step 1



Each Node Has...

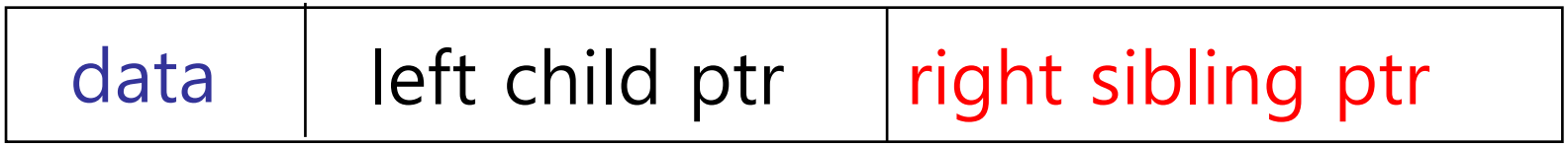
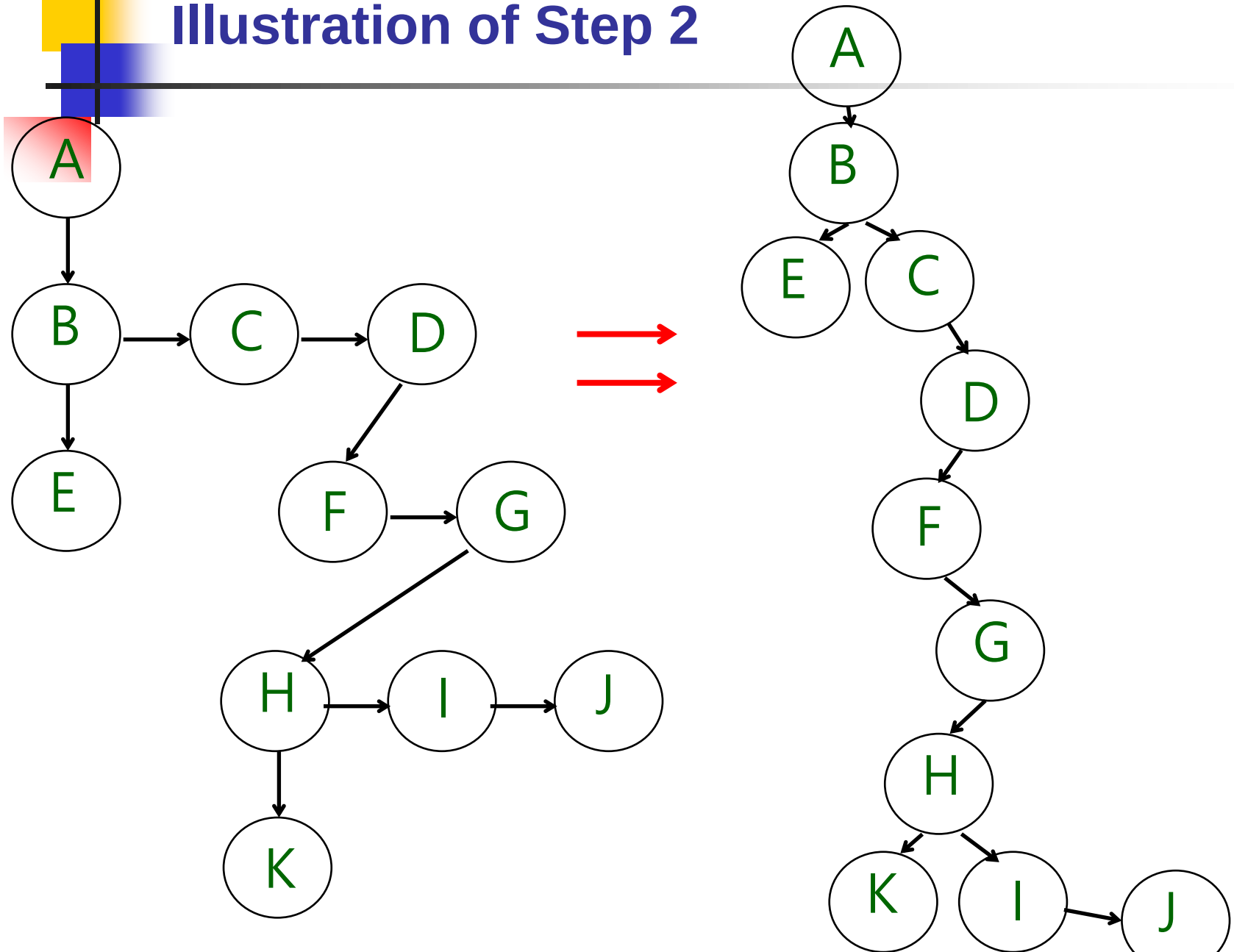


Illustration of Step 2



Each Node Now Has...

Algorithm (1/2)

- The root of the general tree is the root of the binary tree.
- Traverse the general tree in a depth-first order from the root, and make the leftmost child of the node the left child of the corresponding node in the binary tree.
- If the node has no left child, make the right sibling the right child of the node in the binary tree.
- Repeat the above process for the entire general tree.

Algorithm (2/2)

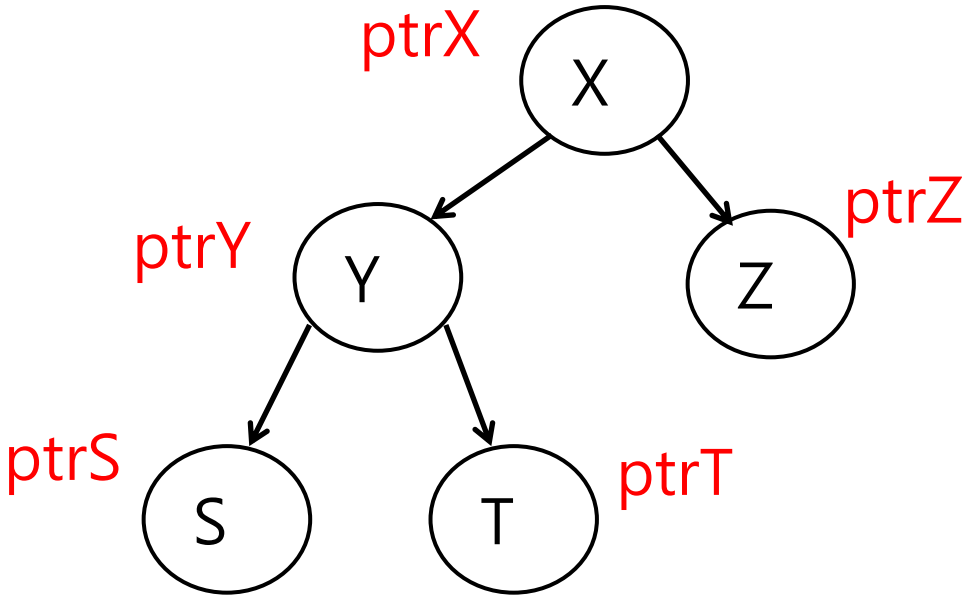
- In the Transformation
  - All nodes of the original general tree are included.
  - Node relative levels are preserved.
  - Parent-descendant relationship is preserved.

Implementing Inorder Traversal in C

```
void inorder (tree_ptr   ptr)
{
     if  (ptr)  {
          inorder (ptr->left_child);
          (visit node);
          inorder (ptr->right_child);
      }
}
```

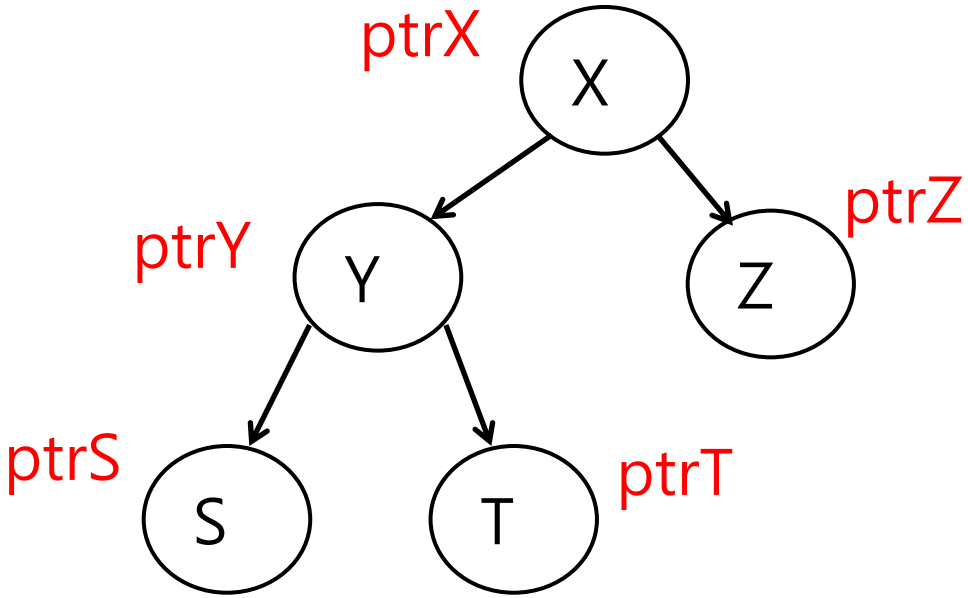
How Recursion Works: (Inorder traversal of a sample tree)

```
void inorder (tree_ptr   ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```



How Recursion Works (execution sequence)

```
void inorder (tree_ptr   ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```

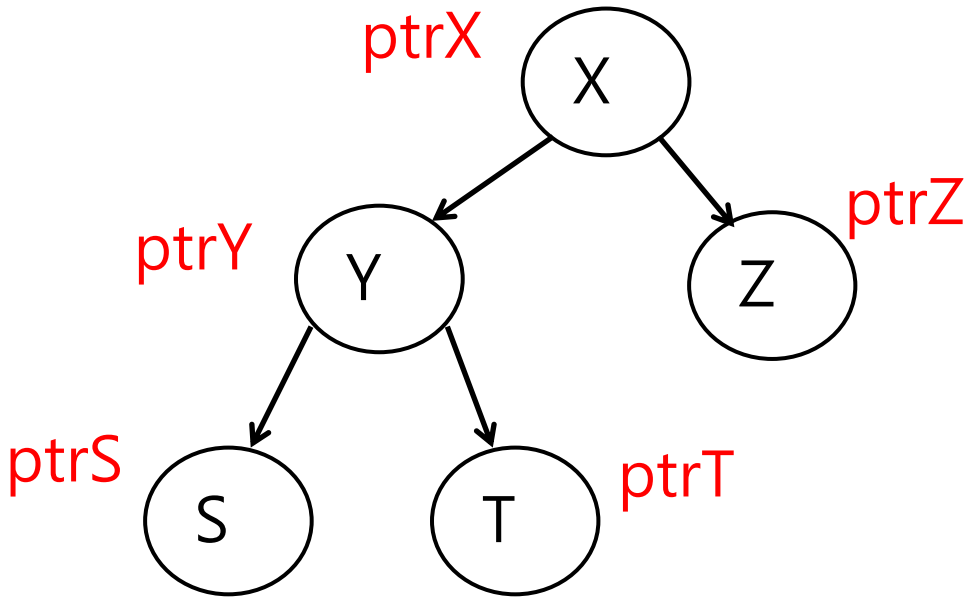


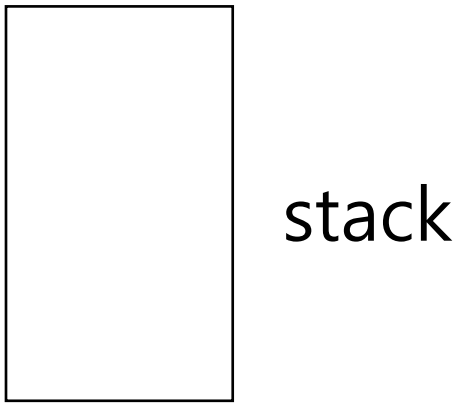

```
inorder (ptrX)
    inorder (ptrY)
        inorder (ptrS)
            inorder (null)
            visit S
            inorder (null)
        visit Y
        inorder (ptrT)
            inorder (null)
            visit T
            inorder (null)
visit X
inorder (ptrZ)
        inorder (null)
        visit Z
        inorder (null)
```

Stack Is Used to Save the Activation Record (* simplified illustration 1/7 *)

```
void inorder (tree_ptr   ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```

inorder (ptrX)



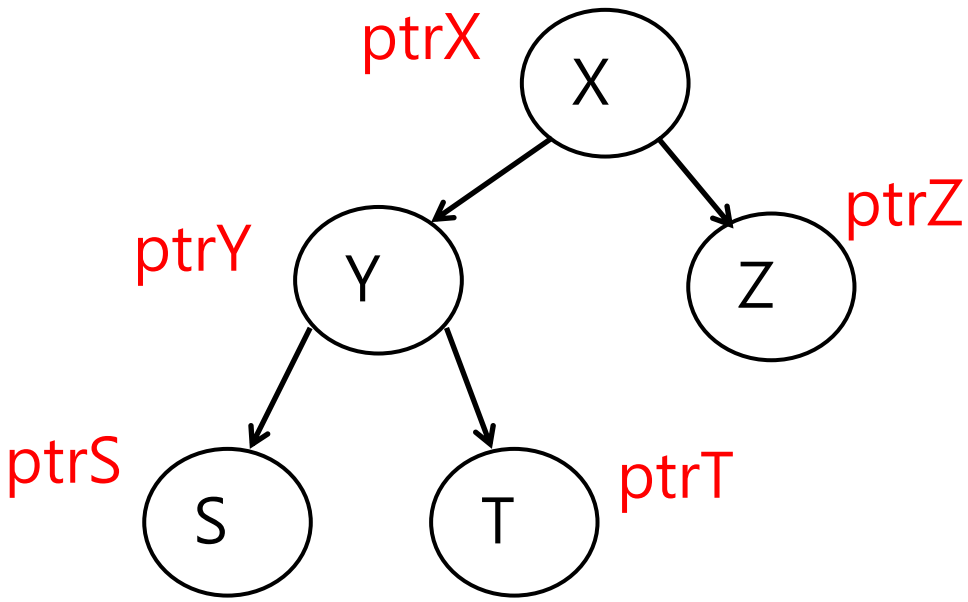


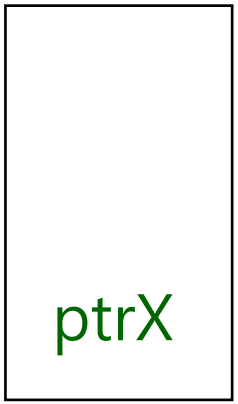
(2/7)

```
void inorder (tree_ptr   ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```

inorder (ptrX)

inorder (ptrY)





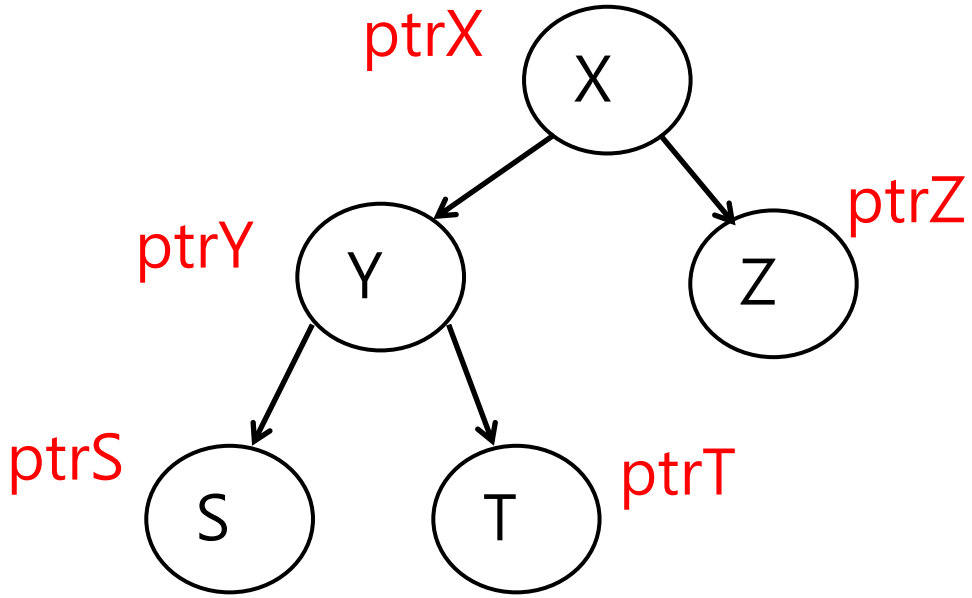
(3/7)

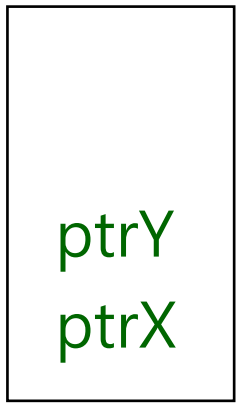
```
void inorder (tree_ptr   ptr)
{ if  (ptr)  {
          inorder (ptr->left_child);
          (visit node);
          inorder (ptr->right_child); }
}
```

inorder (ptrX)

inorder (ptrY)

inorder (ptrS)





(4/7)

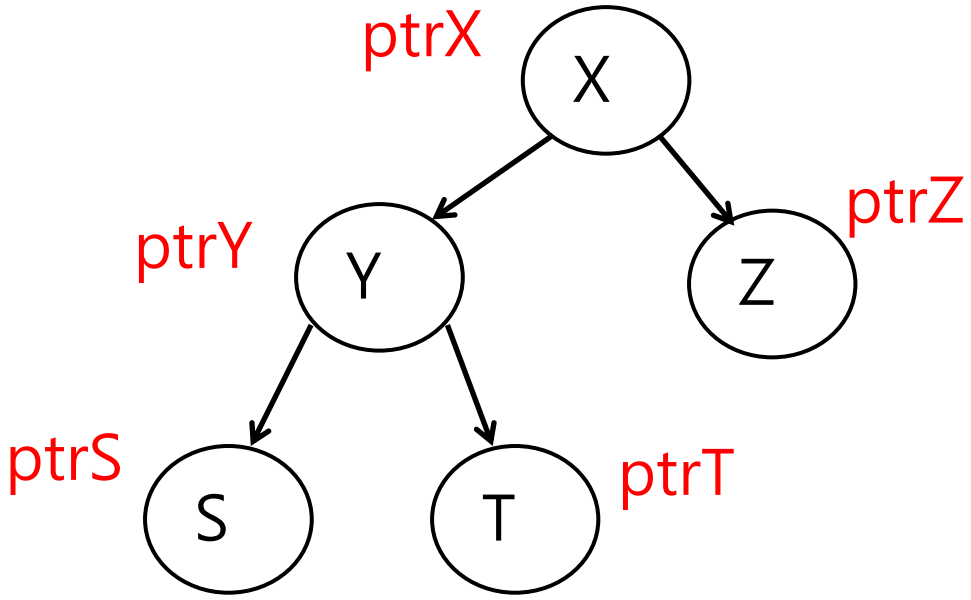
```
void inorder (tree_ptr   ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```

inorder (ptrX)

inorder (ptrY)

inorder (ptrS)

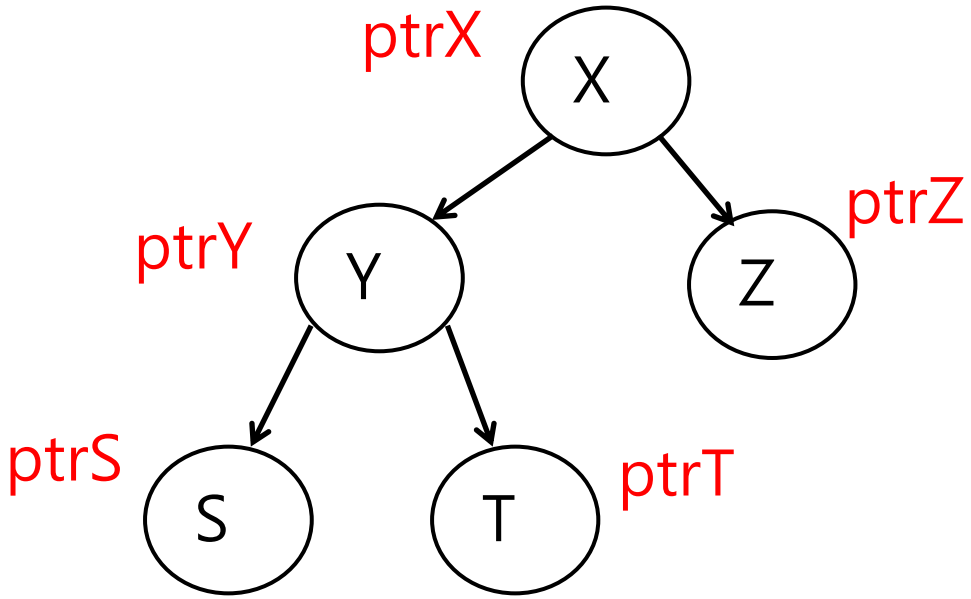
inorder (Lnull)



ptrS
ptrY
ptrX

(5/7)

```
void inorder (tree_ptr   ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```



inorder (ptrX)

inorder (ptrY)

inorder (ptrS)

inorder (Lnull)

visit S

inorder (Rnull)

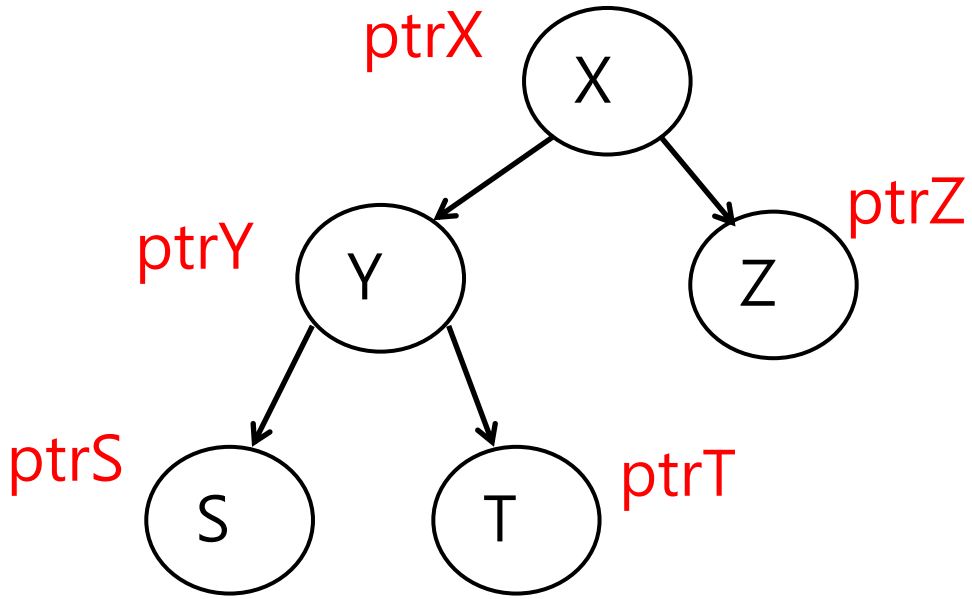
ptrS
ptrY
ptrX

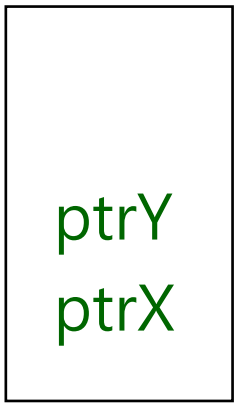
(6/7)

```
void inorder (tree_ptr   ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```

inorder (ptrX)

inorder (ptrY)





(7/7)

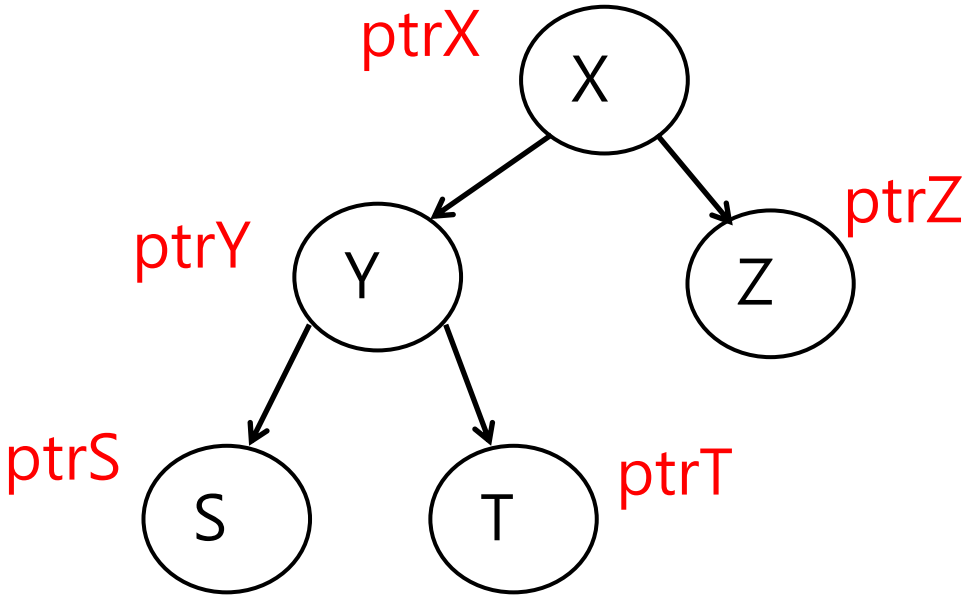
```
void inorder (tree_ptr   ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```

inorder (ptrX)

inorder (ptrY)

visit Y

inorder (ptrT)



ptrT
ptrY
ptrX

** Activation Record (or Stack Frame)

- An activation record is a block of memory used for managing the information needed by a single execution of a program (that includes nested function calls and returns).
- In this lecture, for simplicity, only the function call actual parameter is shown in the stack.
- An activation record actually includes a lot more information.
  - temporary variables, local variables
  - saved machine registers
  - control link, access link
  - (function call) actual parameters, return values

Implementing Postorder Traversal in C

```
void postorder (tree_ptr   ptr)
{
     if  (ptr)  {
          postorder (ptr->left_child);
          postorder (ptr->right_child);
          (visit node);
      }
}
```

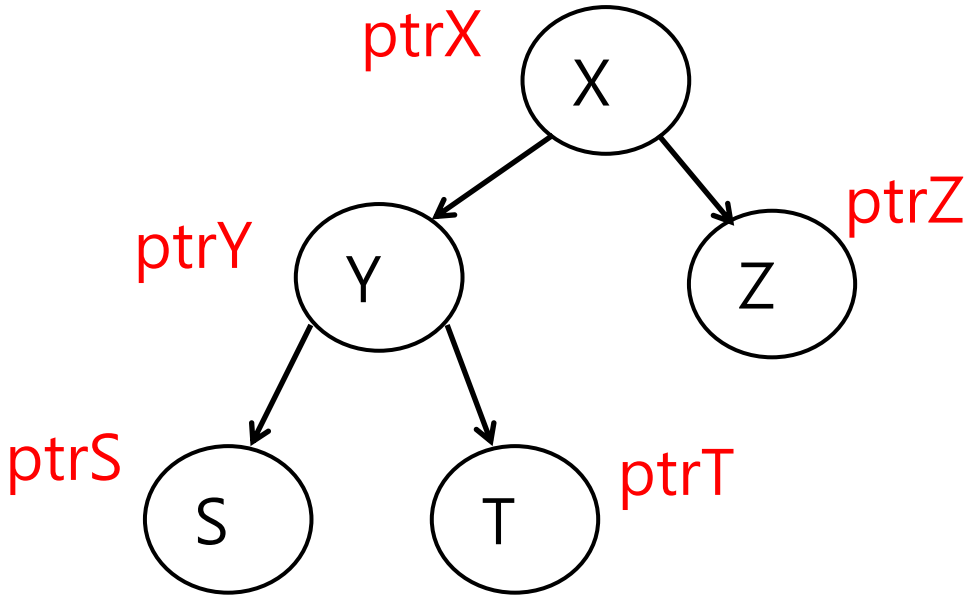
Implementing Preorder Traversal in C

```
void preorder (tree_ptr   ptr)
{
     if  (ptr)  {
          (visit node);
          preorder (ptr->left_child);
          preorder (ptr->right_child);
      }
}
```

Assignment 4

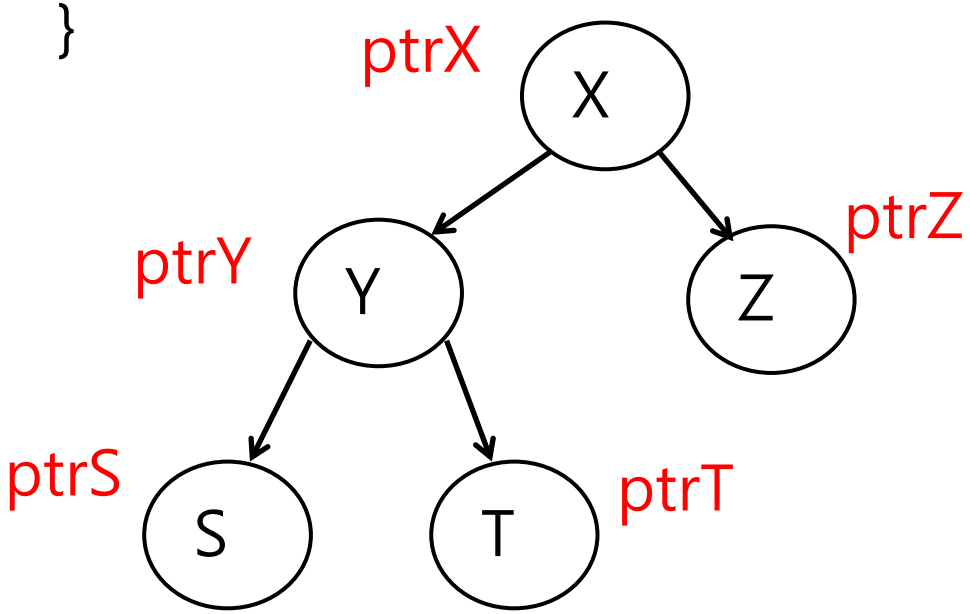
HW 4-1: Draw the Recursion Execution Sequence and Stack State Sequence

```
void postorder (tree_ptr   ptr)
{ if  (ptr)  {
        postorder (ptr->left_child);
        postorder (ptr->right_child);
        (visit node);}
}
```



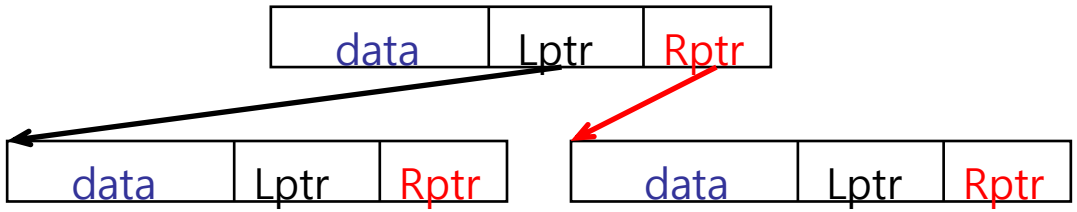
HW 4-2: Draw the Recursion Execution Sequence

```
void preorder (tree_ptr   ptr)
{ if  (ptr)  {
        (visit node);
        preorder (ptr->left_child);
        preorder (ptr->right_child);
        }
}
```

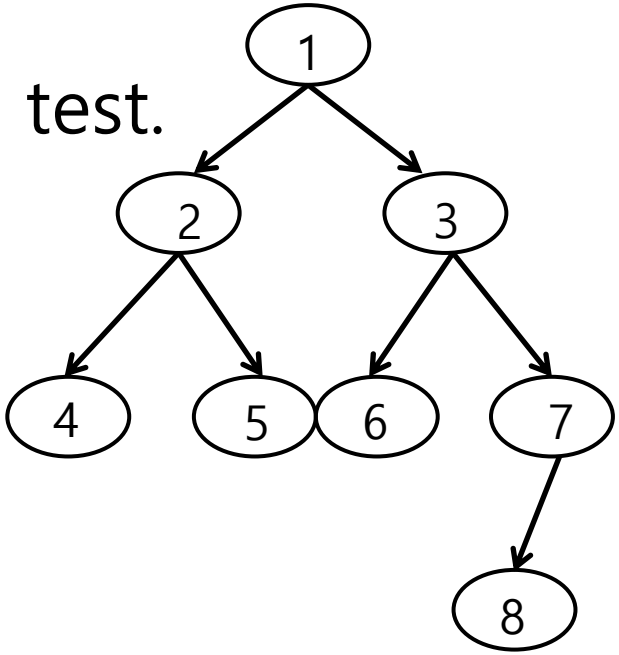


HW 4-3: Implementing a Binary Tree using Linked List

- Implementing a Binary Tree Using a Singly Linked List With 2 Pointers



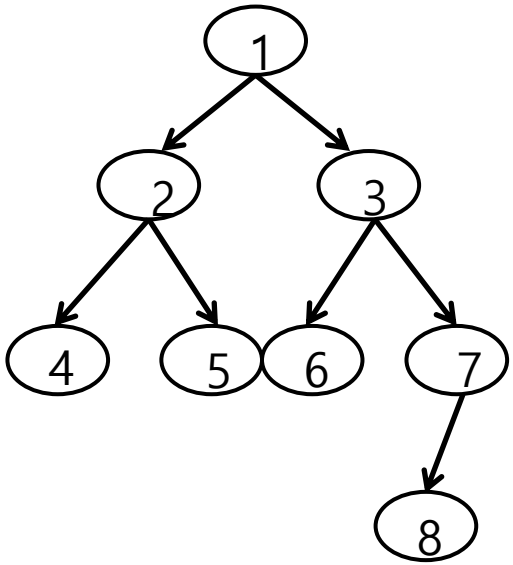
- Use this tree for test.

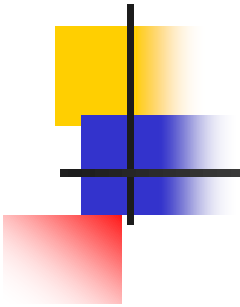


HW 4-4: Implementing Binary Tree Traversal Methods

- Implement #1. Preorder, #2. Inorder, #3. Postorder traversal methods.

- Test with this tree.
  - When a node is visited, printout the key.





End of Lecture