Data Structures:
Trees: Introduction, Tree Traversal, Binary Trees, Binary Expression Trees

Won Kim
(Lecture by Youngmin Oh)
Spring 2022

Trees

Real Tree

leaves

root

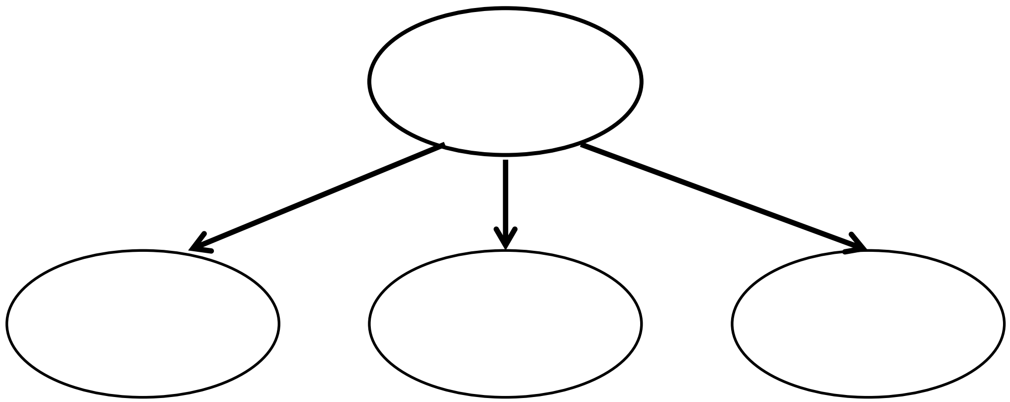
Definitions for Trees in Software Data Structures

Tree Data Structure

root

leaves

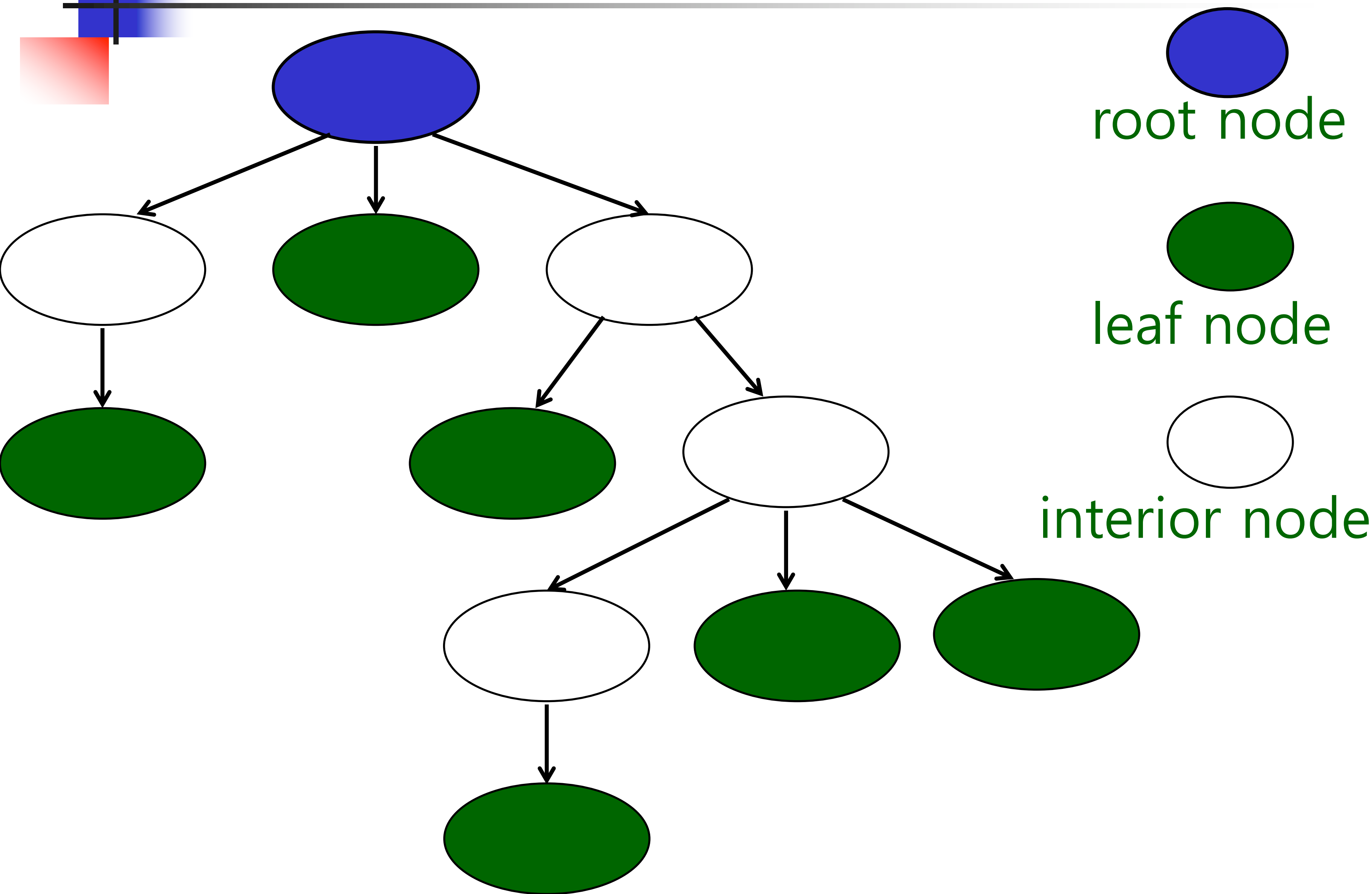
Nodes and Branches (Links)



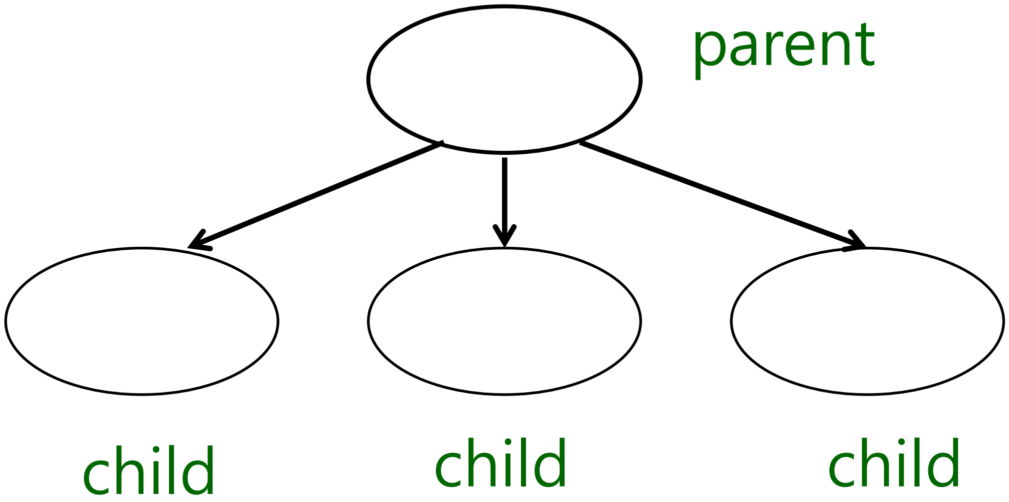
node (stores data)

branch (link)
(pointer stored in a node)

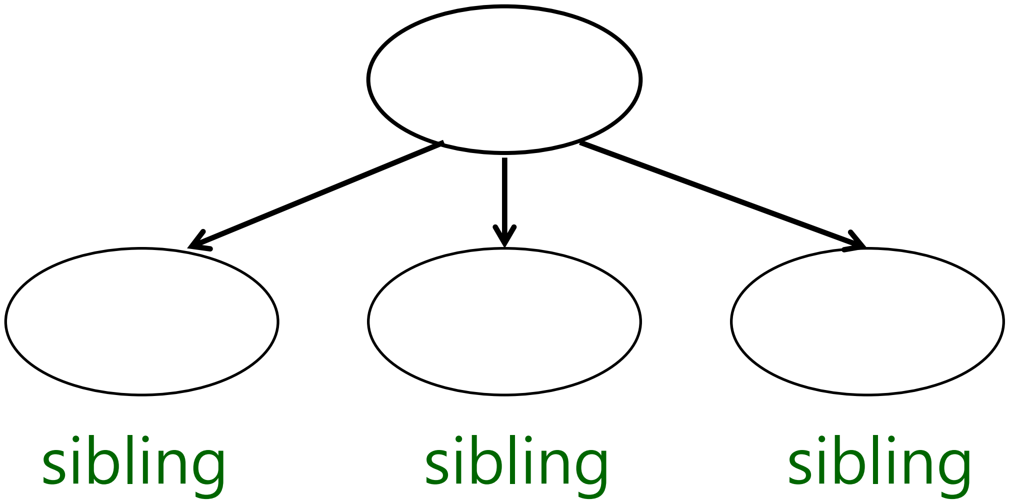
Root, Leaf, Interior (Non-Leaf) Nodes



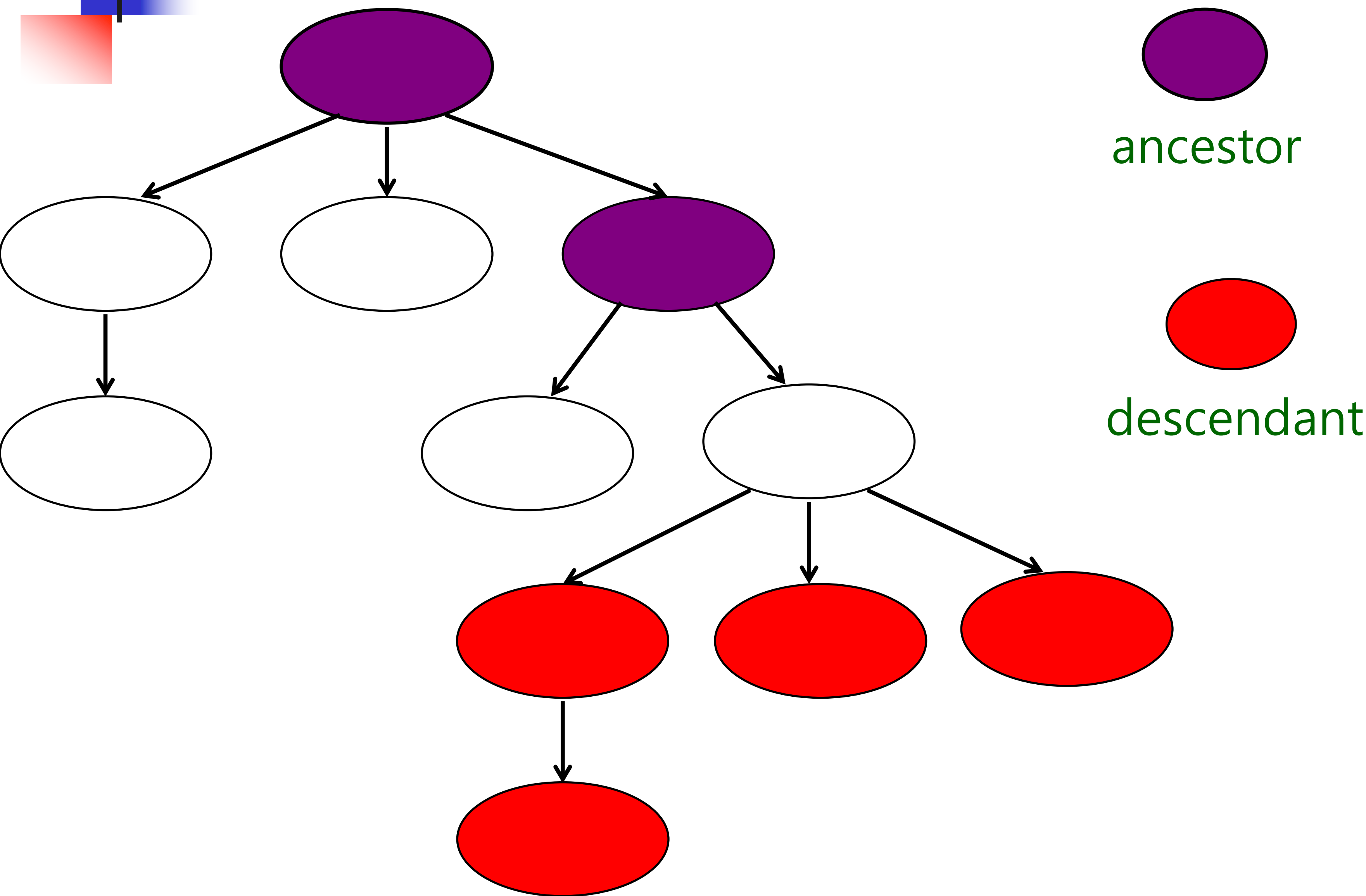
Parent and Child



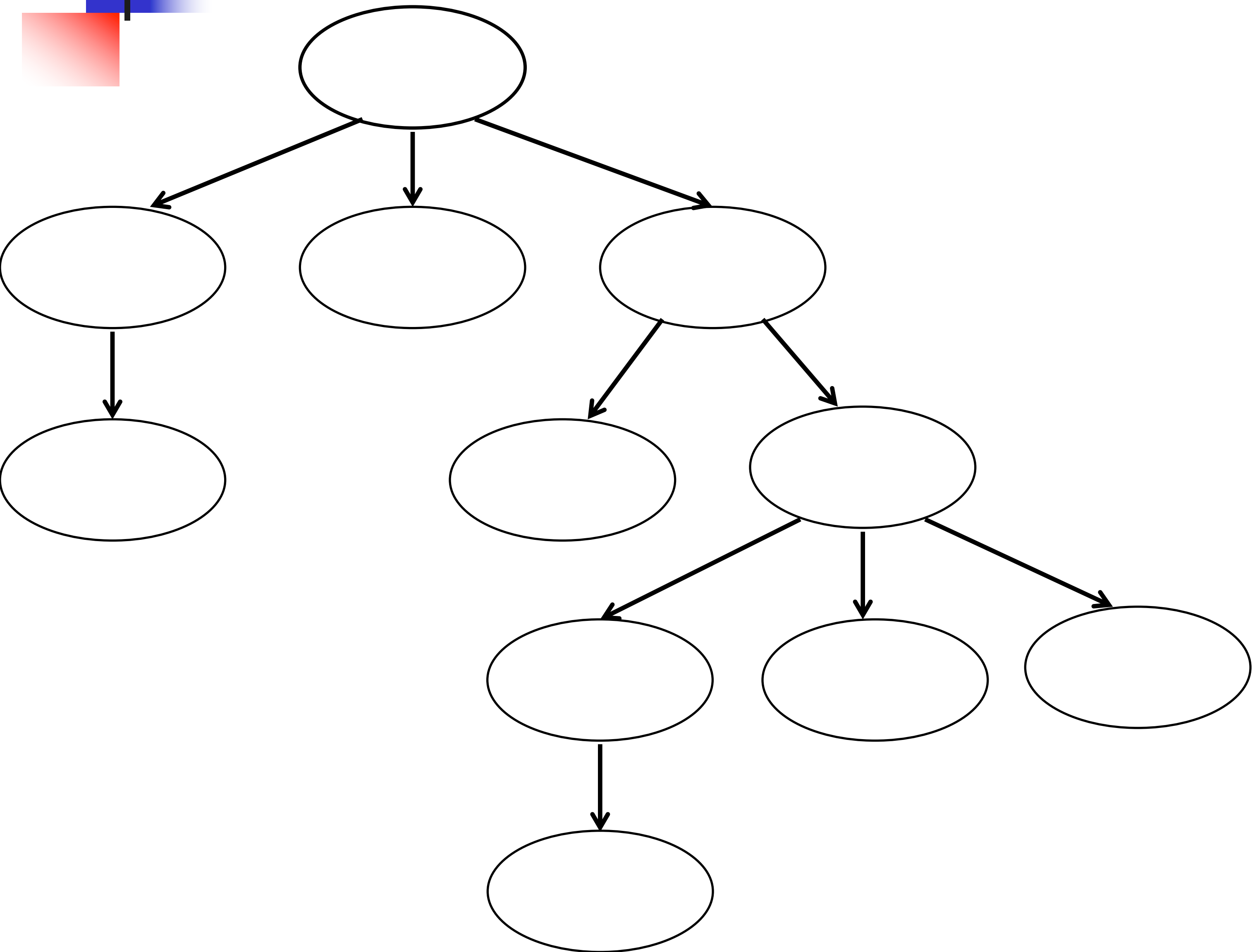
Siblings (brothers and sisters)



Ancestors and Descendants



Depth (Height, Level)



1 or 0

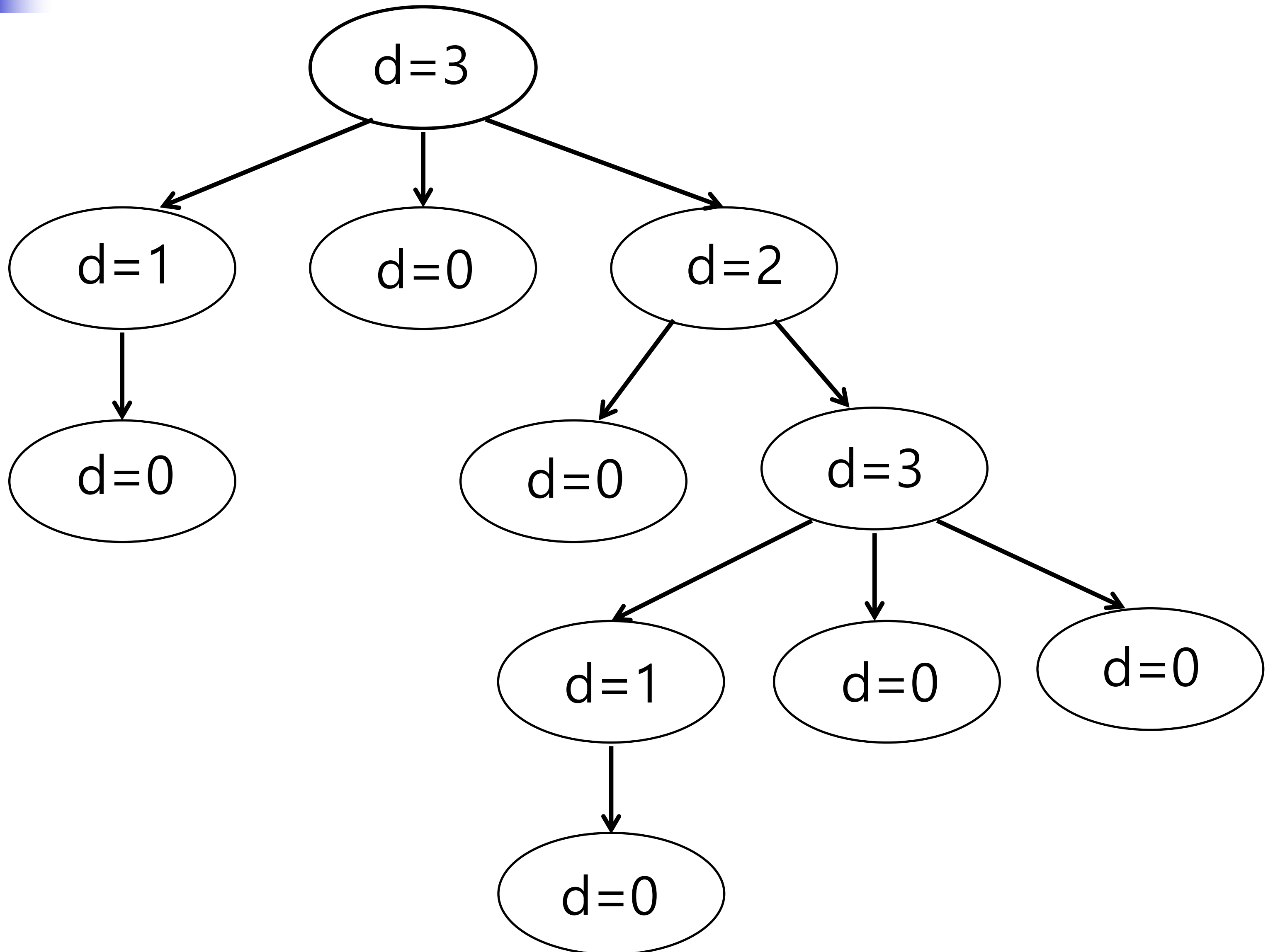
2 or 1

3 or 2

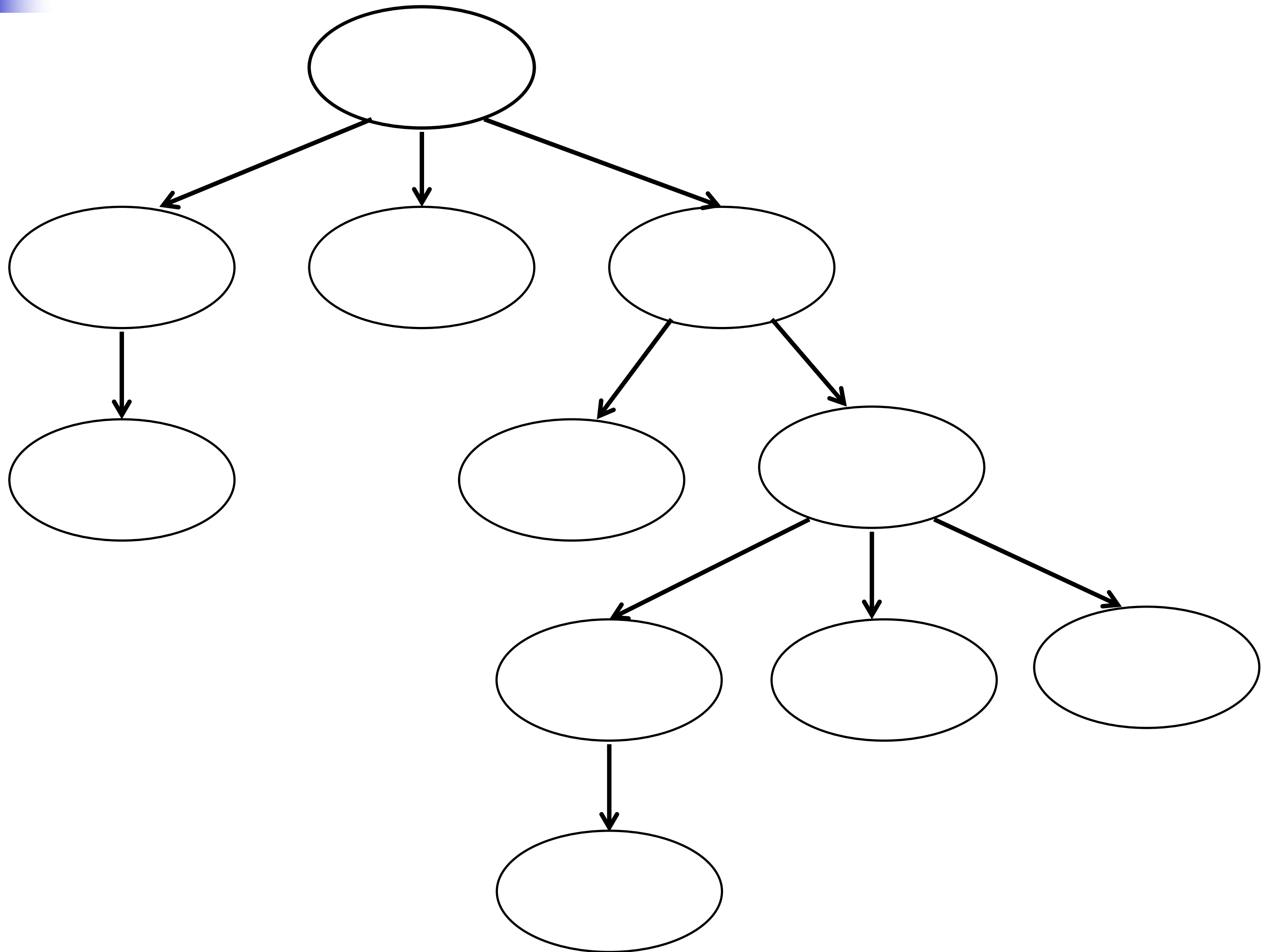
4 or 3

5 or 4

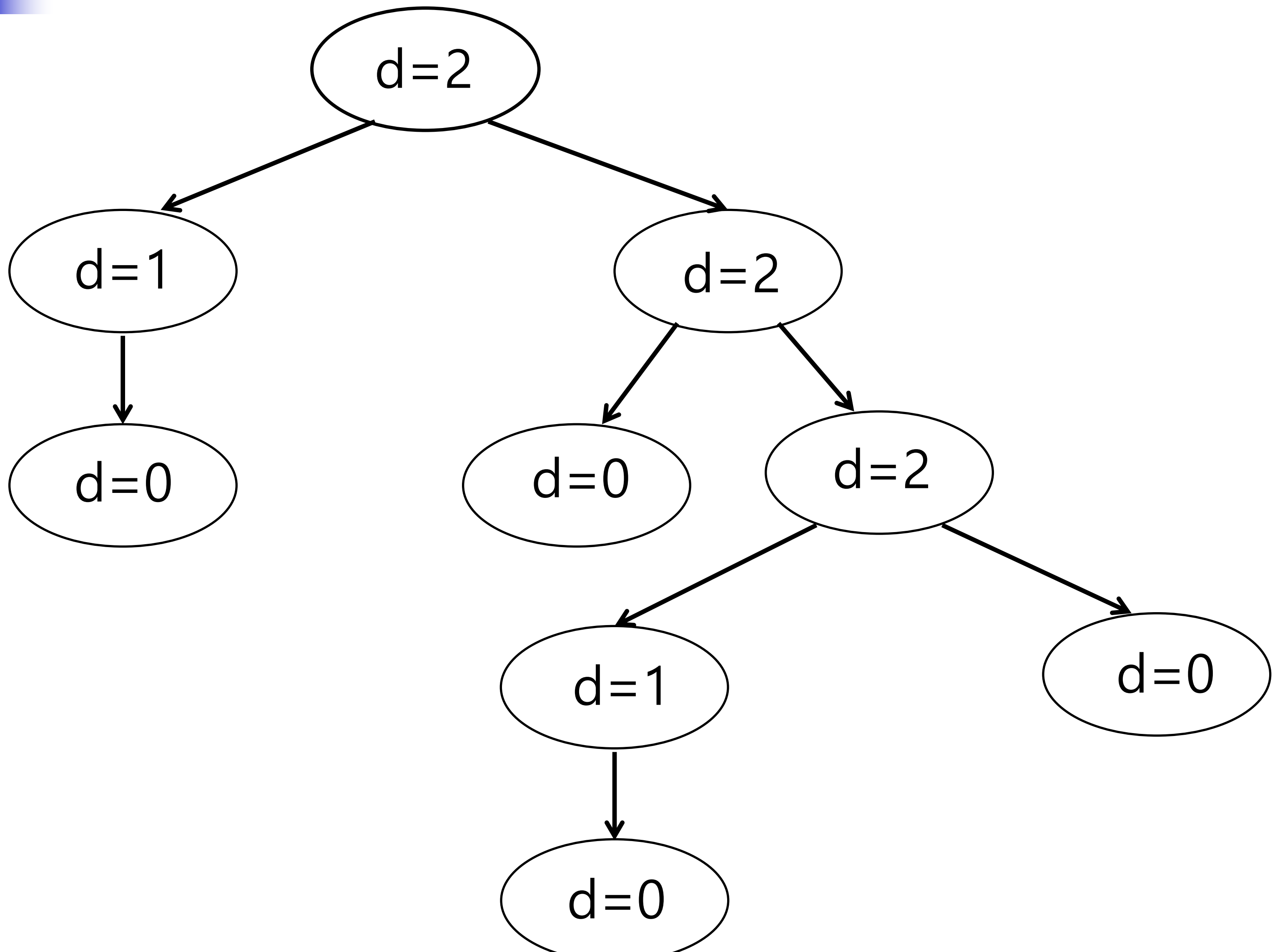
Degree (Fanout)



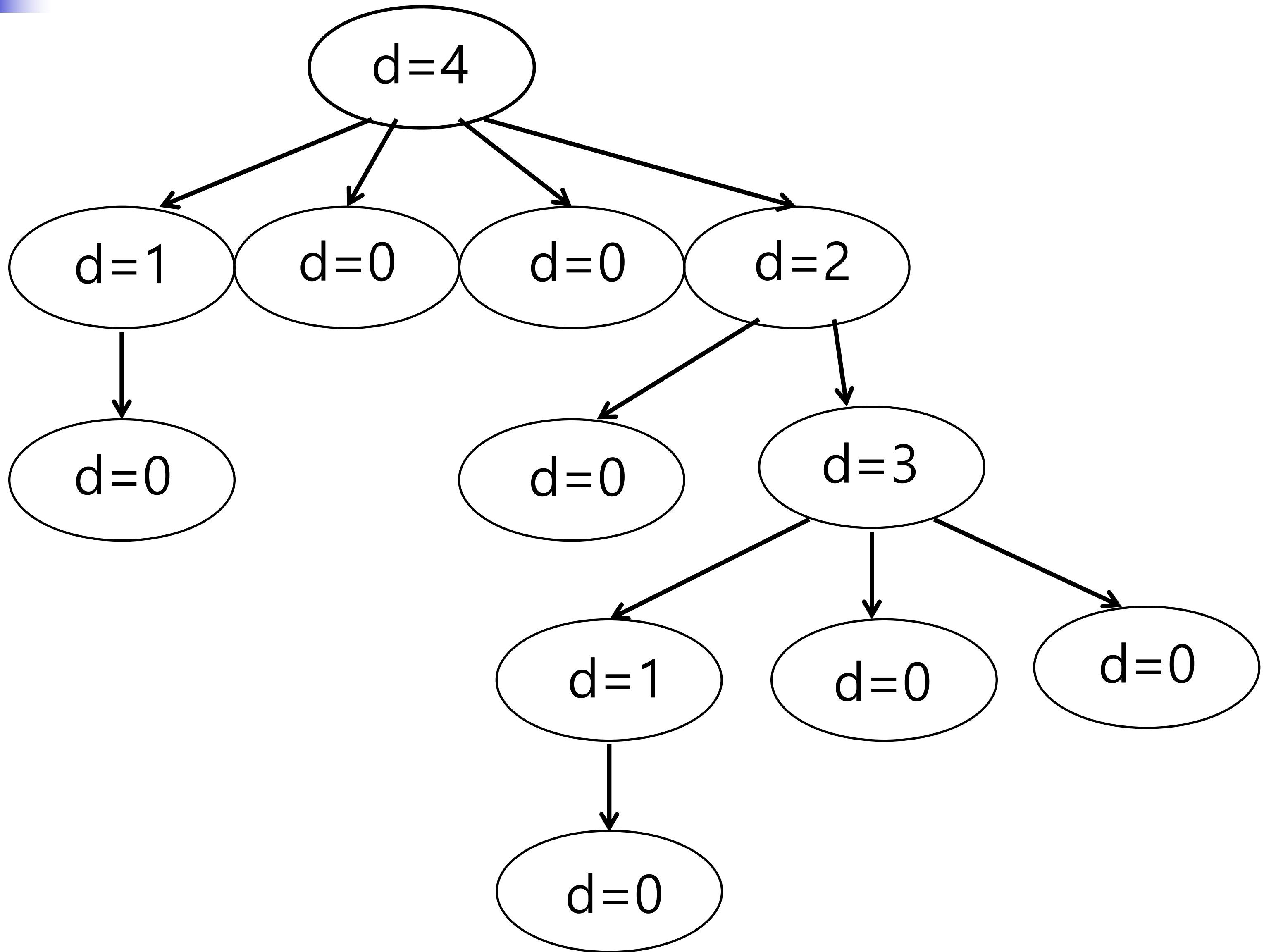
A General Tree



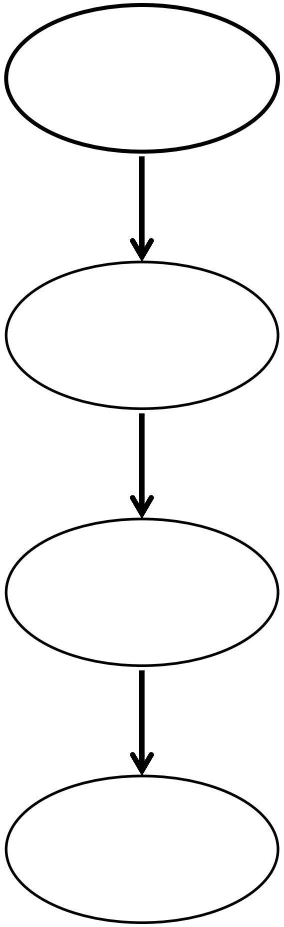
A Binary Tree (Degree <= 2)



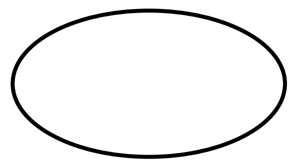
A Quad Tree (Degree =< 4)



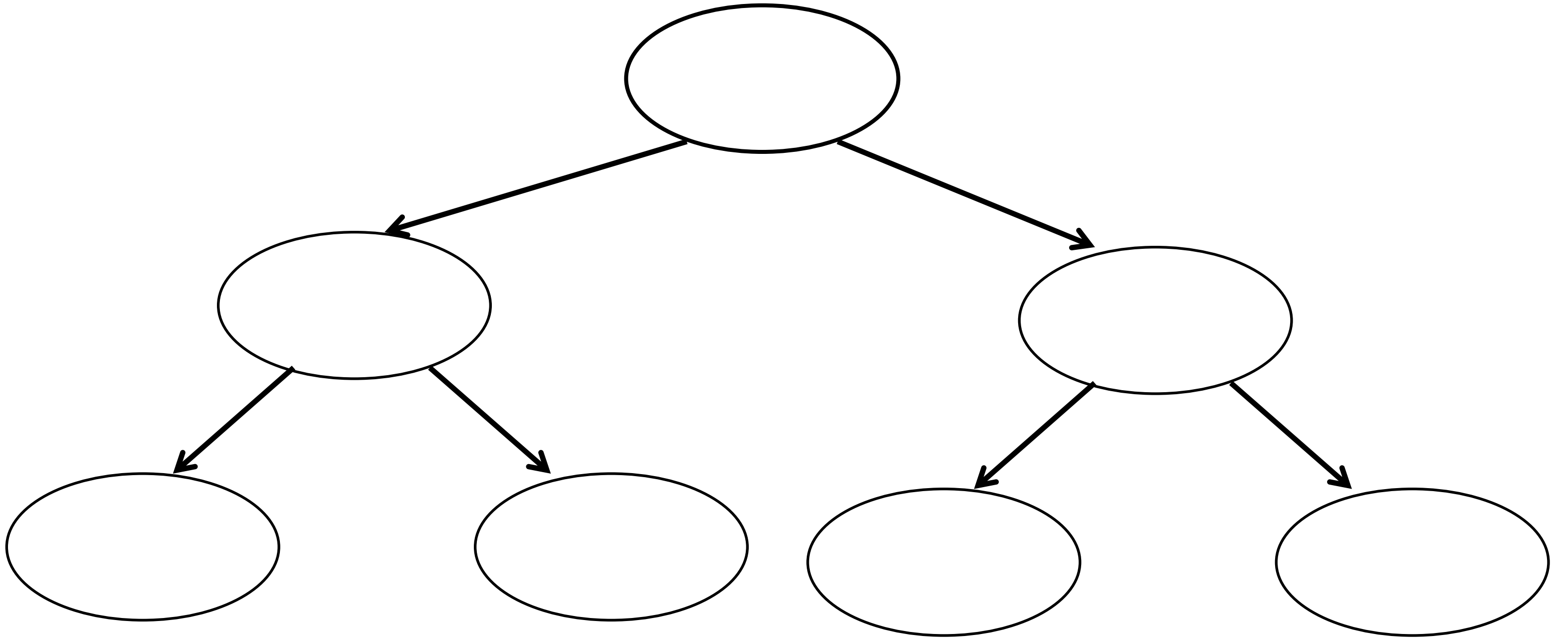
A Skewed Tree (Degenerate Tree)



(Theoretically A) Tree



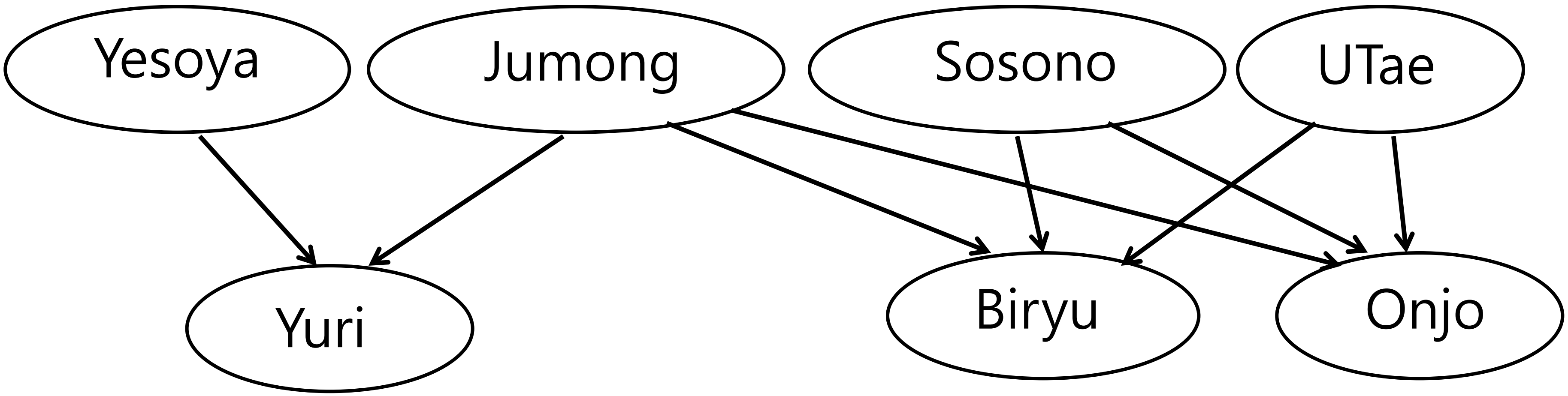
A Full (Perfect) Tree



Organization

- Each non-root node has only one parent node.
  - The root node has no parent node.
- Each non-leaf node has one or more child nodes.
  - Each leaf node has zero child node.
- A tree consists of
  - The root node, and j child nodes of the root node.
  - Each of the j child nodes is the root node of a tree.

Not a Tree: General Genealogy



Many Types of Tree Data Structures
(* We will learn about the red highlighted trees in this course.)

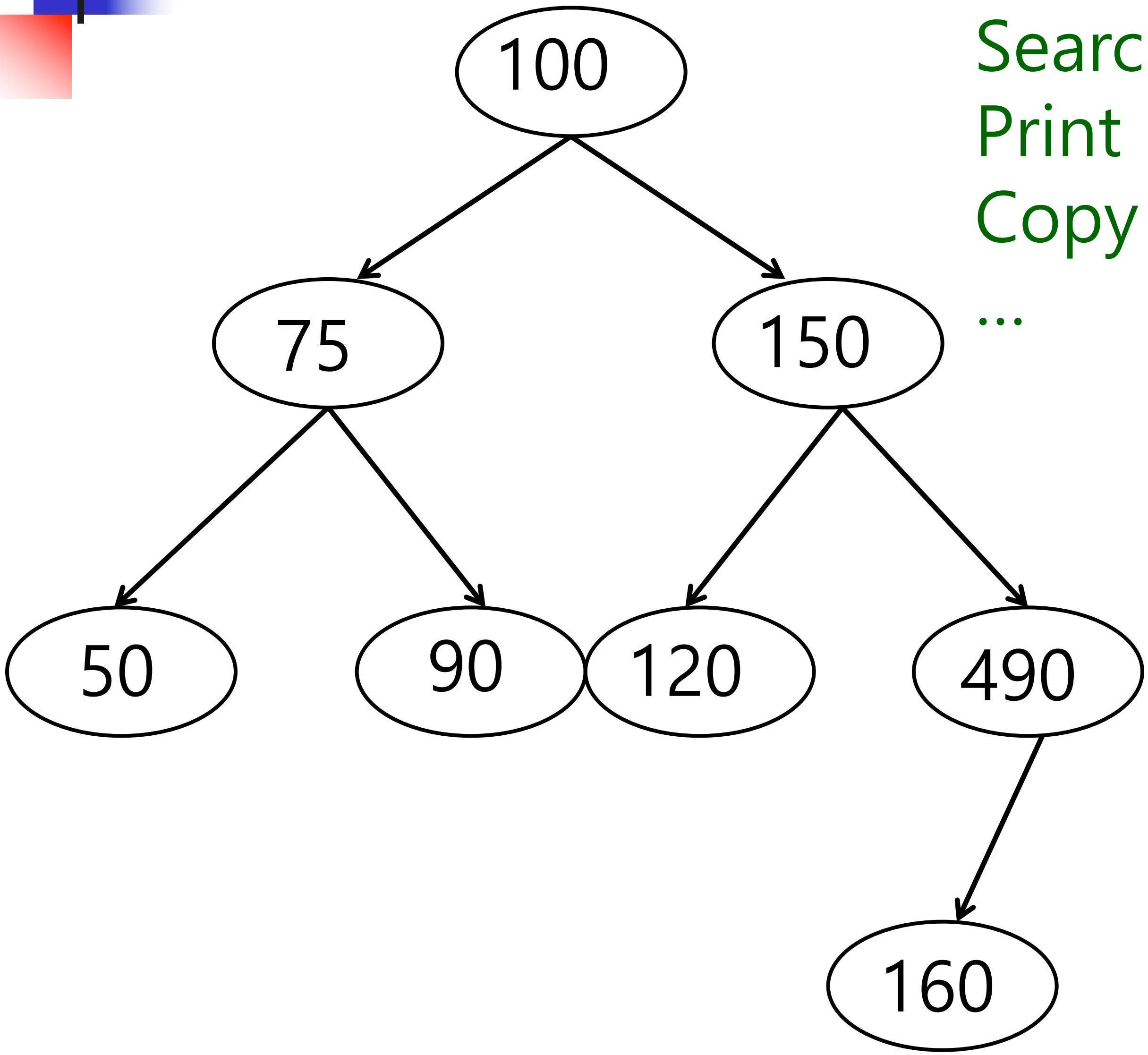
- Binary Tree
  - Binary Search Tree, Heap, Digital Search Tree, Trie
  - Red-Black Tree, AA Tree, Splay Tree,
- Height-Balanced Binary Trees
  - AVL Tree, T-Tree
- n-Way Tree
  - m-Way Trie
  - 2-3 Tree, 2-3-4 Tree
- Height-Balanced m-Way Tree
  - B-Tree, B+-Tree, K-d B-Tree
- Spatial Tree
  - Quad Tree, Oct Tree, K-d Tree, R-Tree, R* Tree

Summary Definition of a Tree

- A tree is a linked list of data, where a data item is linked to other data items by a certain relationship.
- A search for a data item on a tree proceeds from one data item to another data item that satisfies a certain relationship.
- The hierarchy of a tree is a convenient visualization of the relationships among the data items.

Tree Traversal

Tree Traversal (Tree Walk)



Search for a Key
Print Each Key
Copy Each Key
…

Two Basic Ways to Visit Each Node Once

- Visiting a Node: Taking Some Action on the Node
  - printing the data, pushing the data onto a stack, copying data into another tree,…
- Depth-First (Traversal / Search)
- Breadth-First (Traversal / Search)

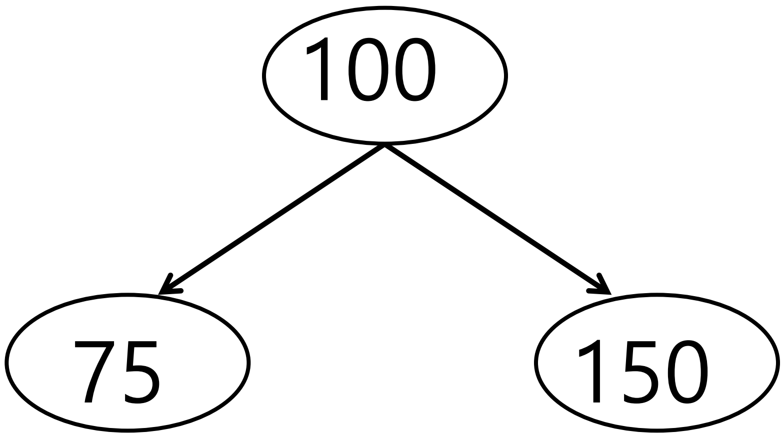
Breadth-First Traversal / Search

- **Level Order Traversal**
  - **Visit every node of a level, and move to the next level.**

Depth-First Traversal / Search

- Inorder Traversal
  - (Left Subtree, Visit the Root, Right Subtree)
- Postorder Traversal
  - (Left Subtree, Right Subtree, Visit the Root)
- Preorder Traversal
  - (Visit the Root, Left Subtree, Right Subtree)

Depth-First Traversal: Exercise



How to Memorize the 3 Types of Depth-First Traversal?

Think of a Mother with Two Children

Quiz: What are the 3 ways to share some cookies among the 3 people?

- Assumption: Mother likes the left child more.
- Your answers??

Answers

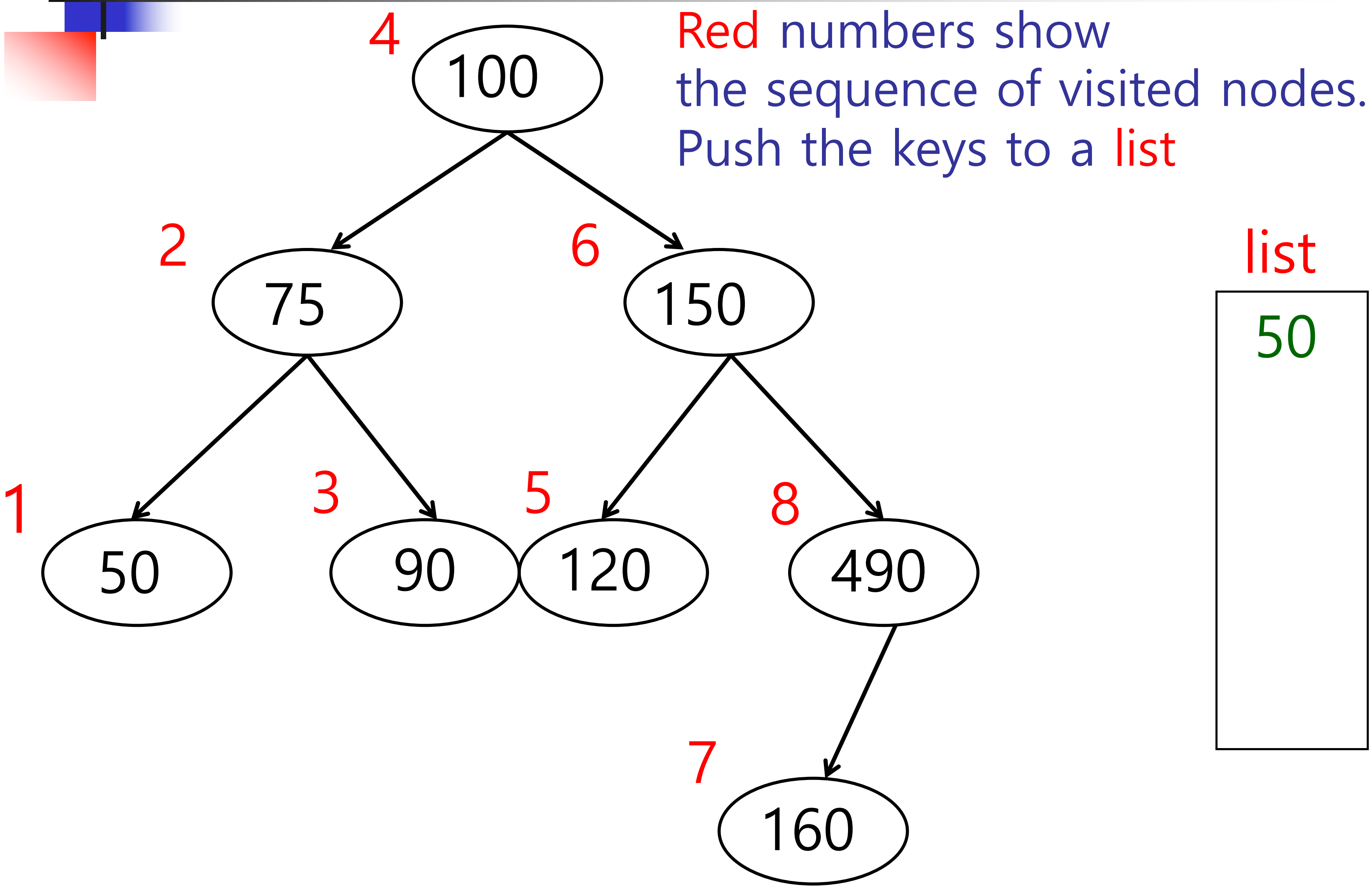
- left child first, mother next, right child last
  - (Inorder)
- left child first, right child next, mother last
  - Unselfish mother (postorder)
- mother first, left child next, right child last
  - Selfish mother, (preorder)

Inorder Traversal

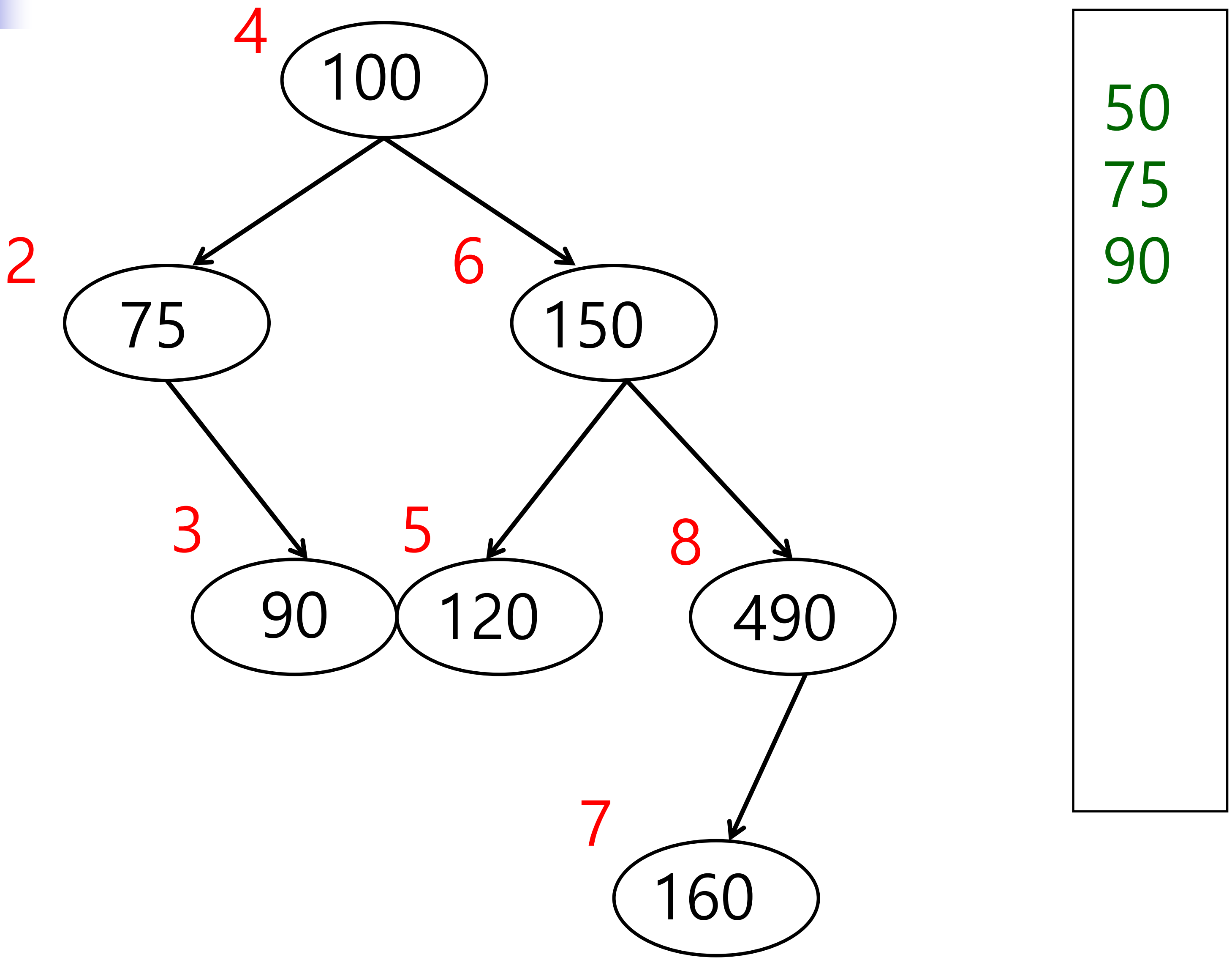
- First Child, Mother, Second Child
  - print or push the key of the visited node to a stack
- Applications
  - retrieval of a sorted sequence
- Makes Sense Only for a Binary Tree

Inorder Traversal: Example (1/6)

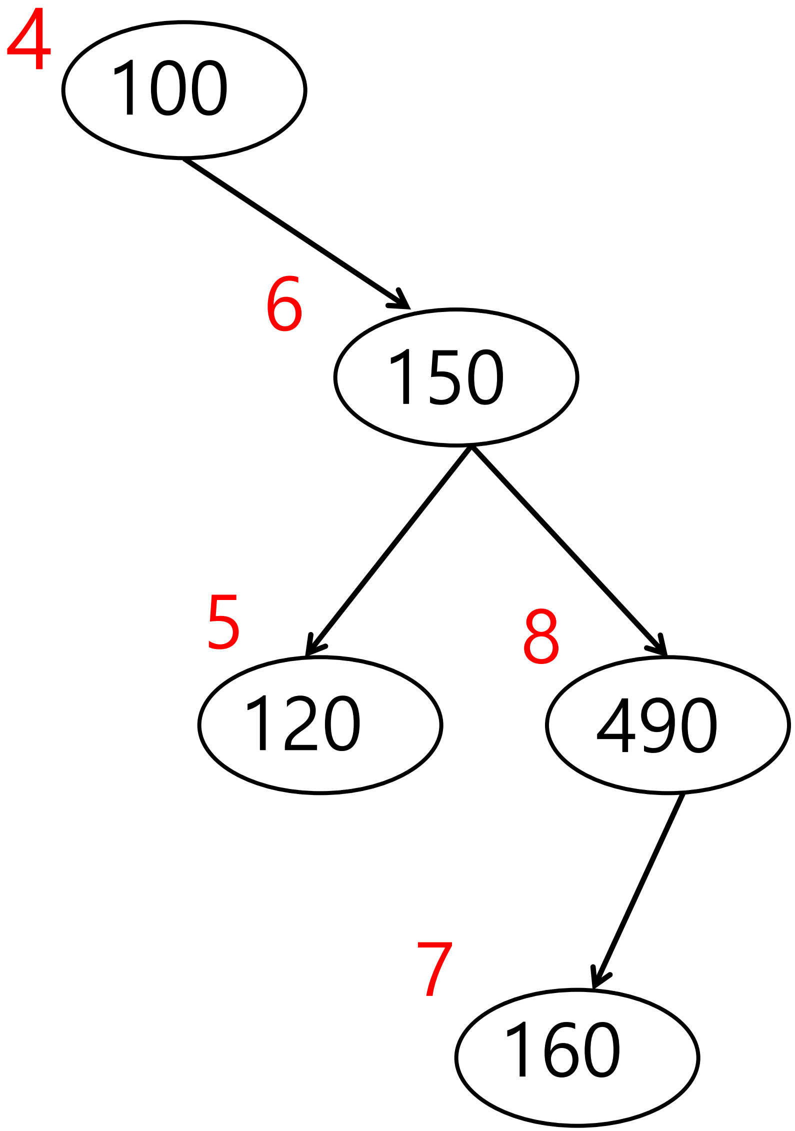
Red numbers show
the sequence of visited nodes.
Push the keys to a list

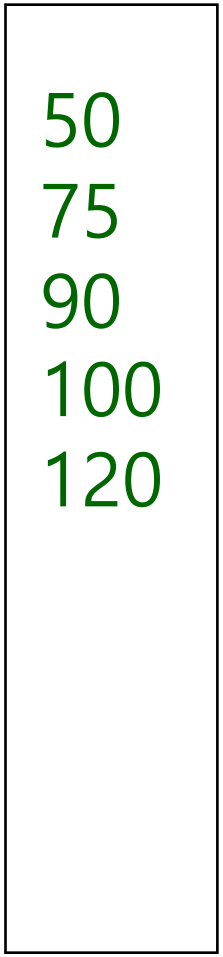


Inorder Traversal: Example (2/6)

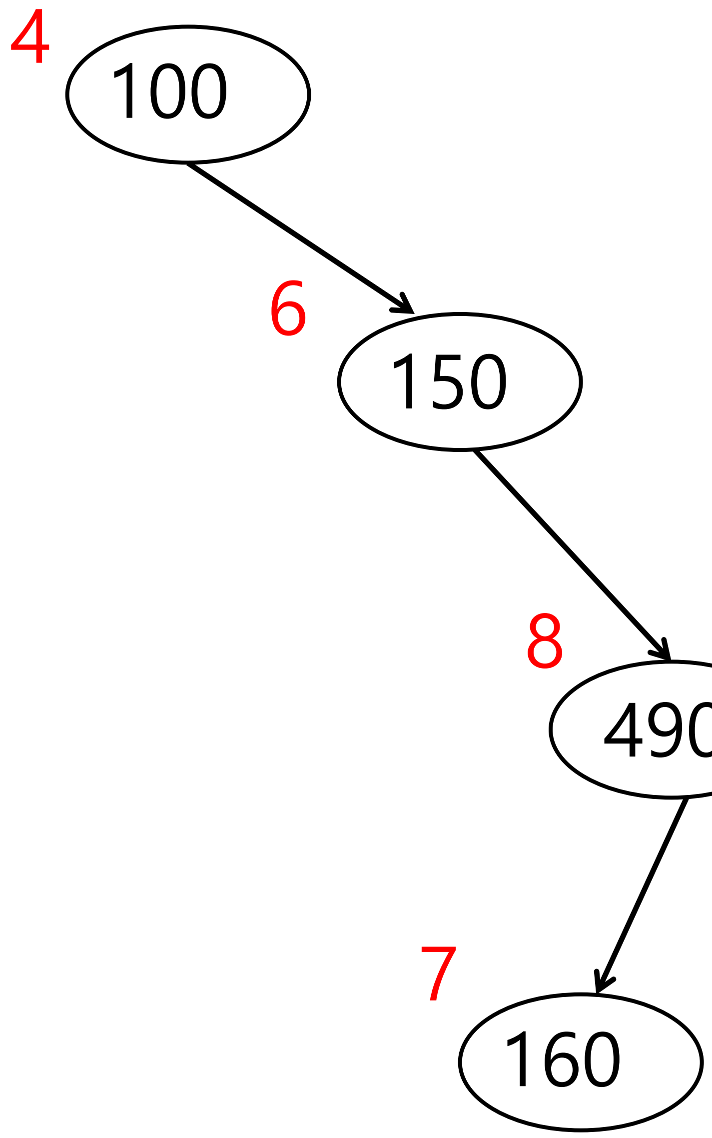


Inorder Traversal: Example (3/6)



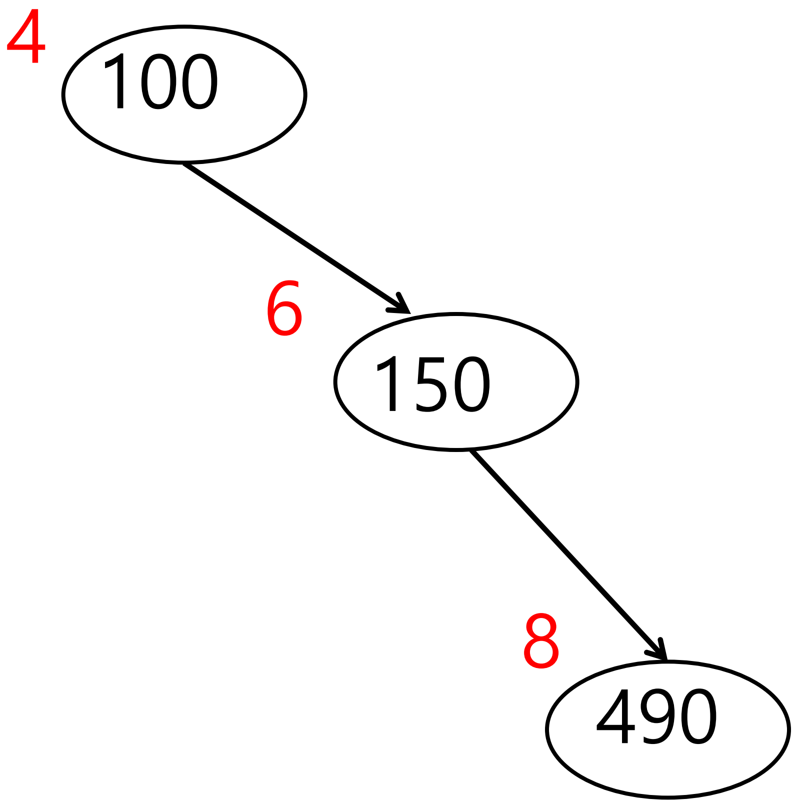


Inorder Traversal: Example (4/6)



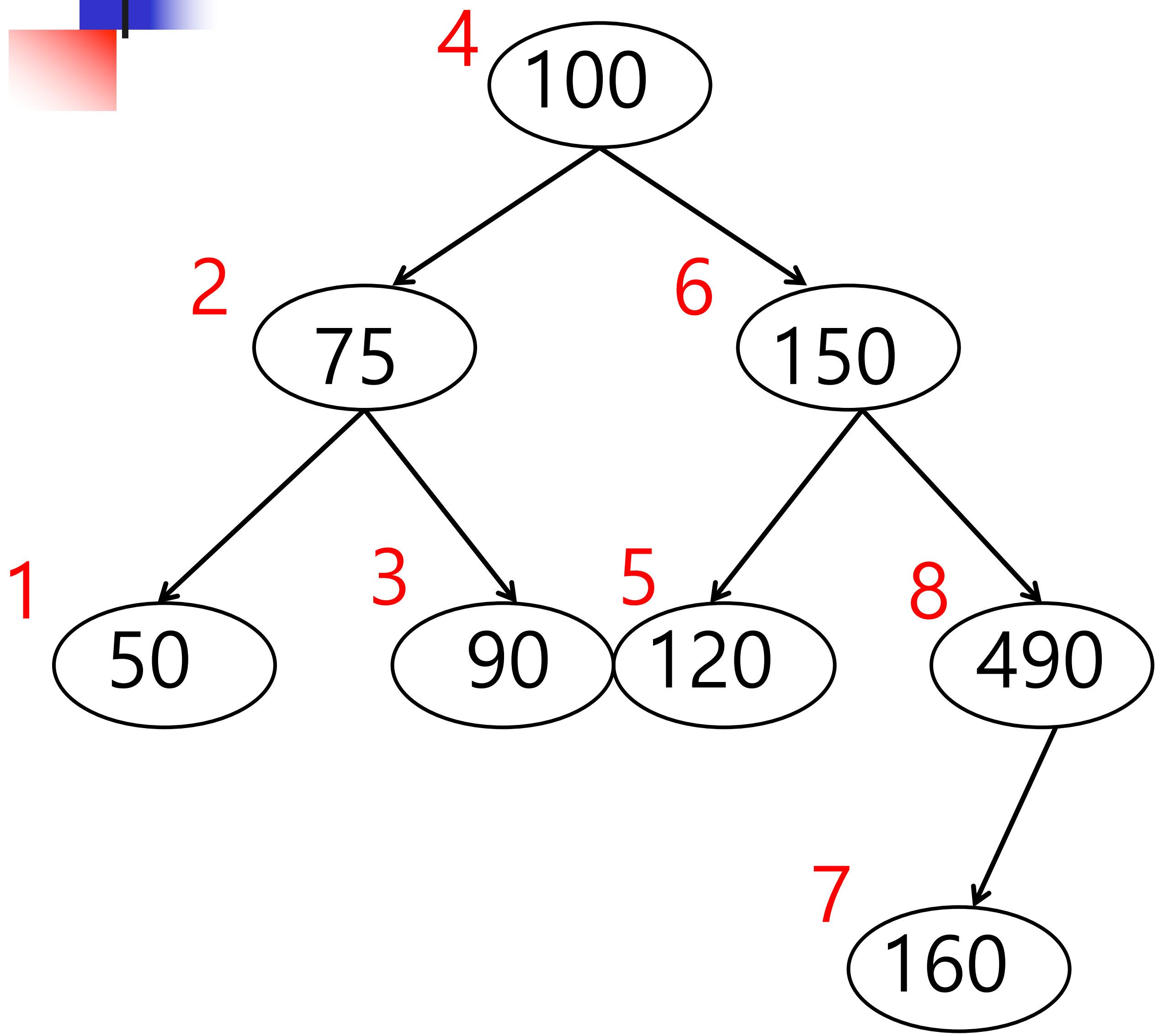
50
75
90
100
120
150
160

Inorder Traversal: Example (5/6)



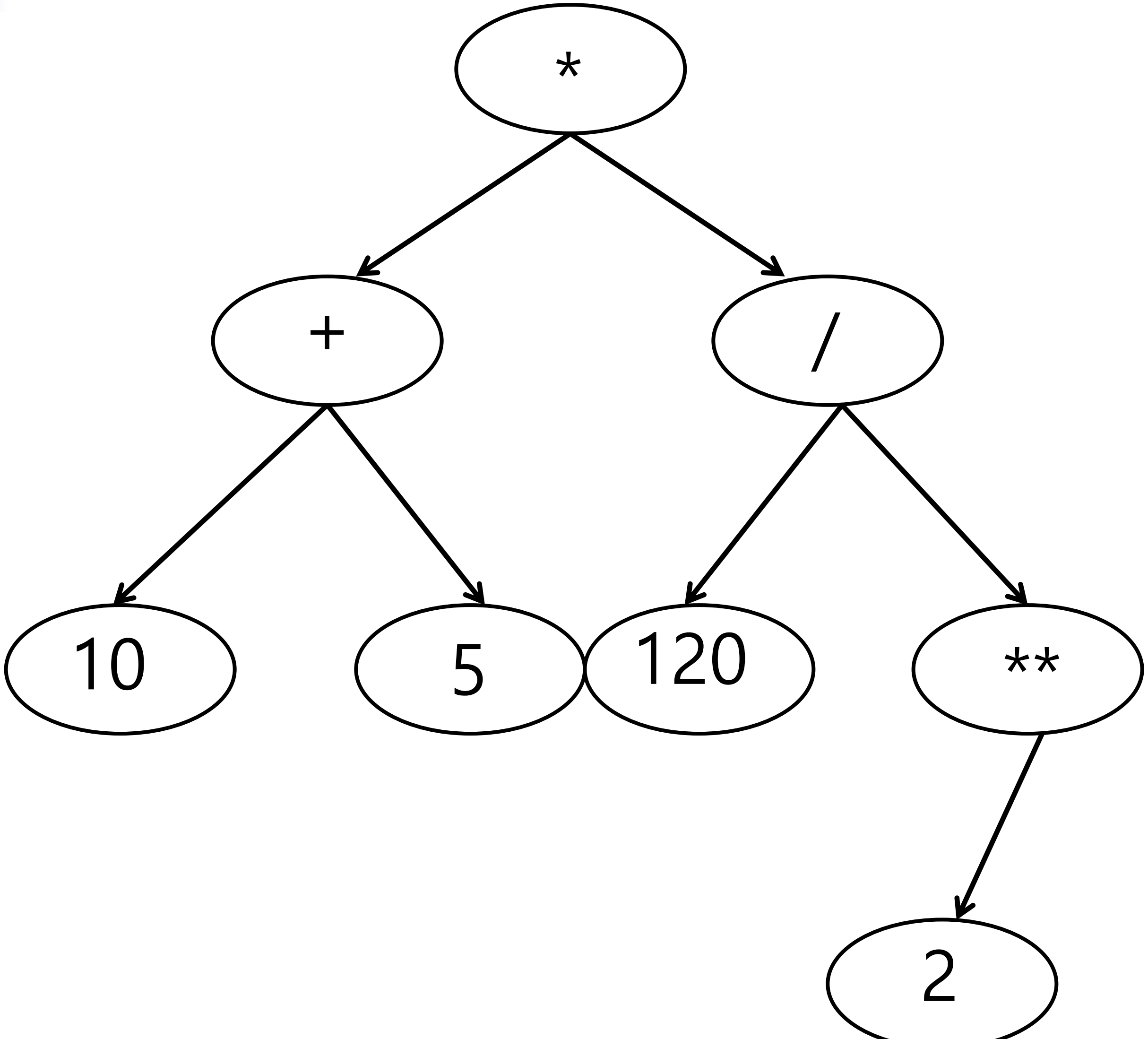
50
75
90
100
120
150
160
490

Inorder Traversal: Example (6/6)

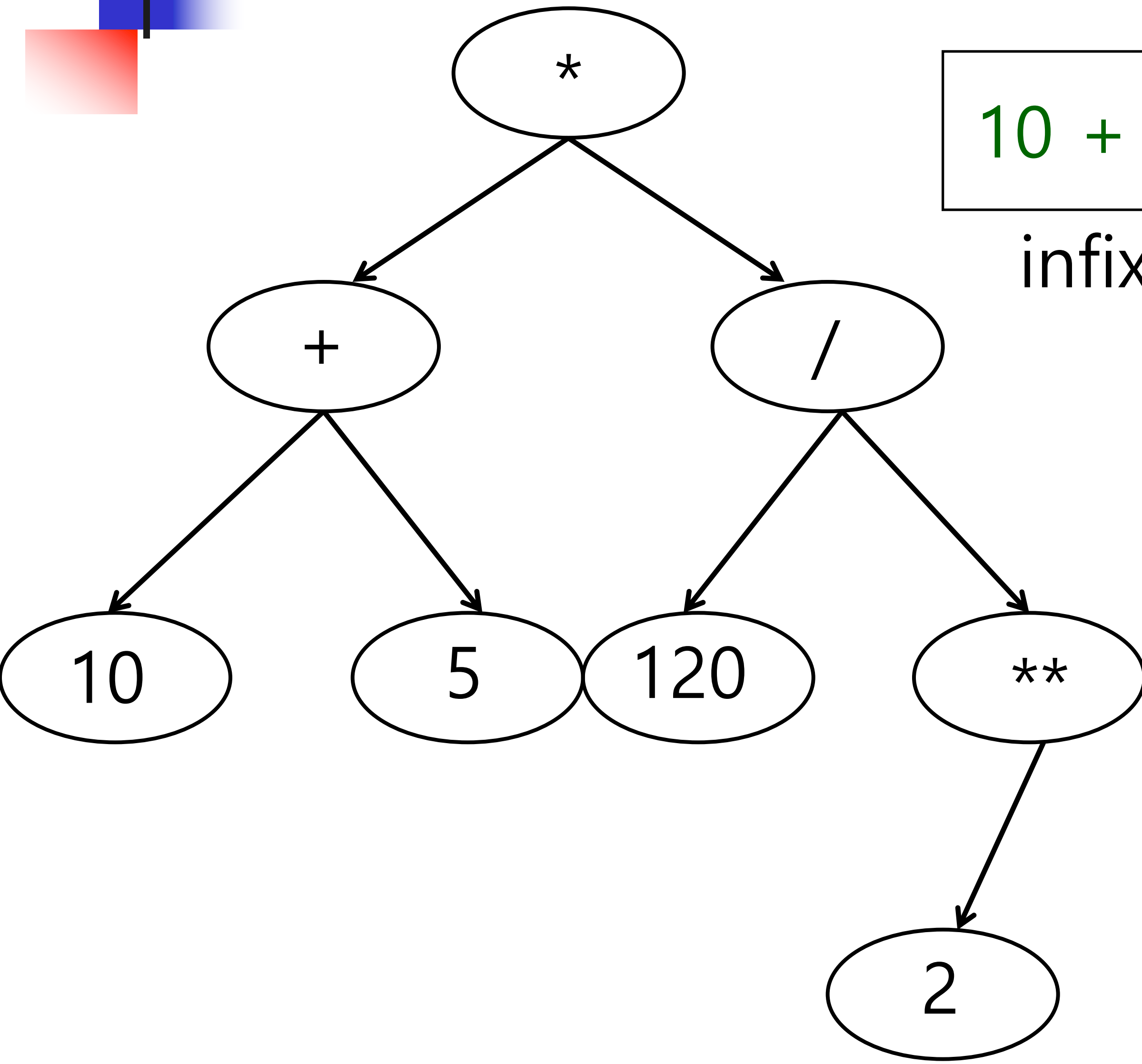


50
75
90
100
120
150
160
490

Inorder Traversal: Exercise



Inorder Traversal: Solution



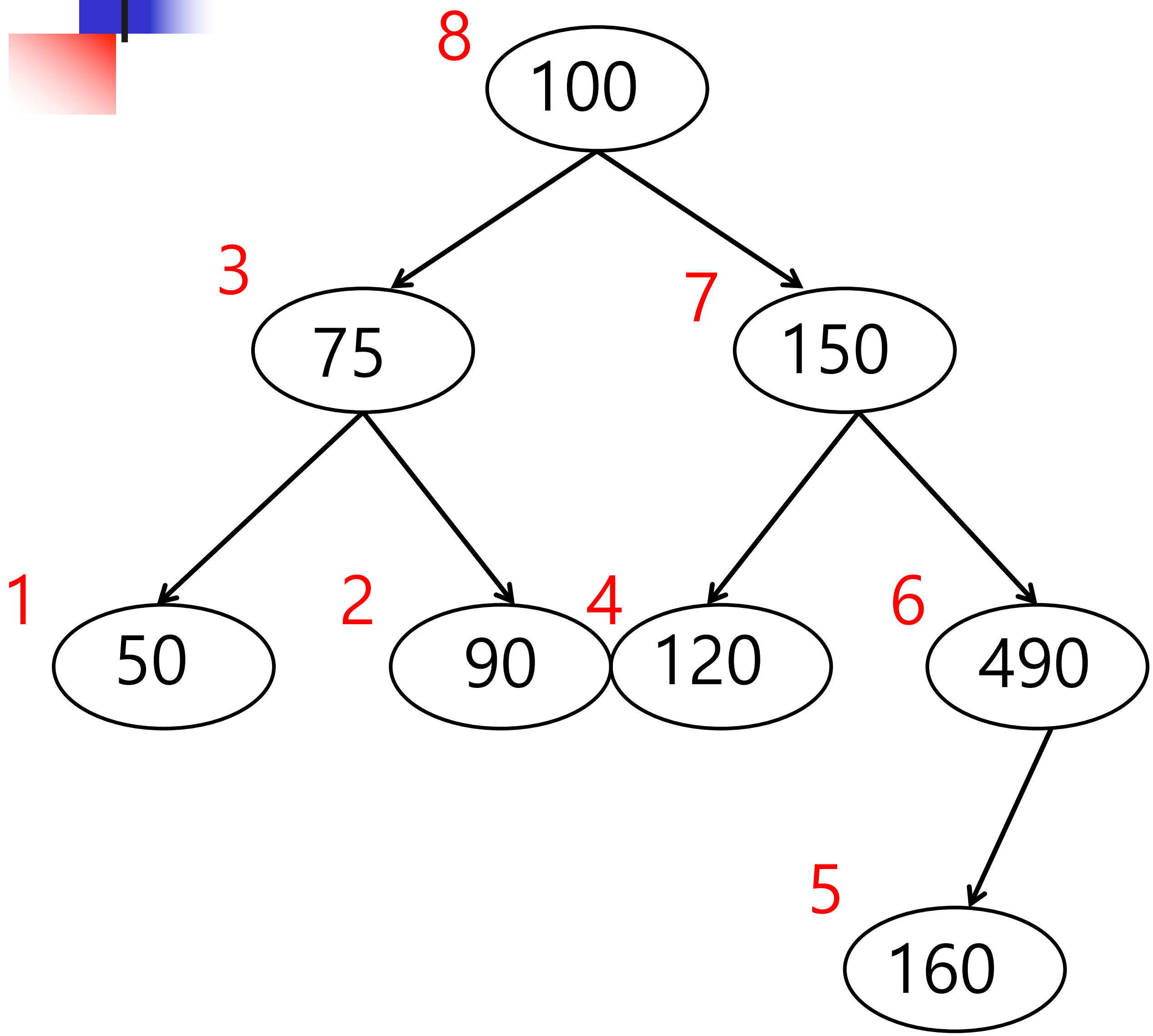
10 + 5 * 120 / 2 **

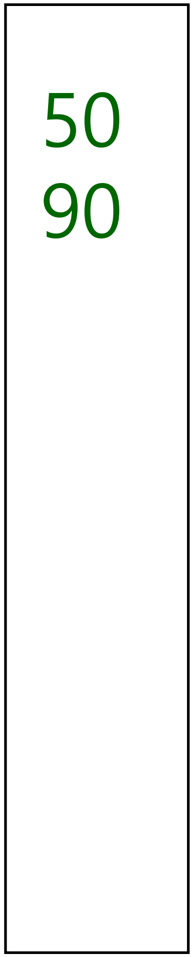
infix expression

Postorder Traversal

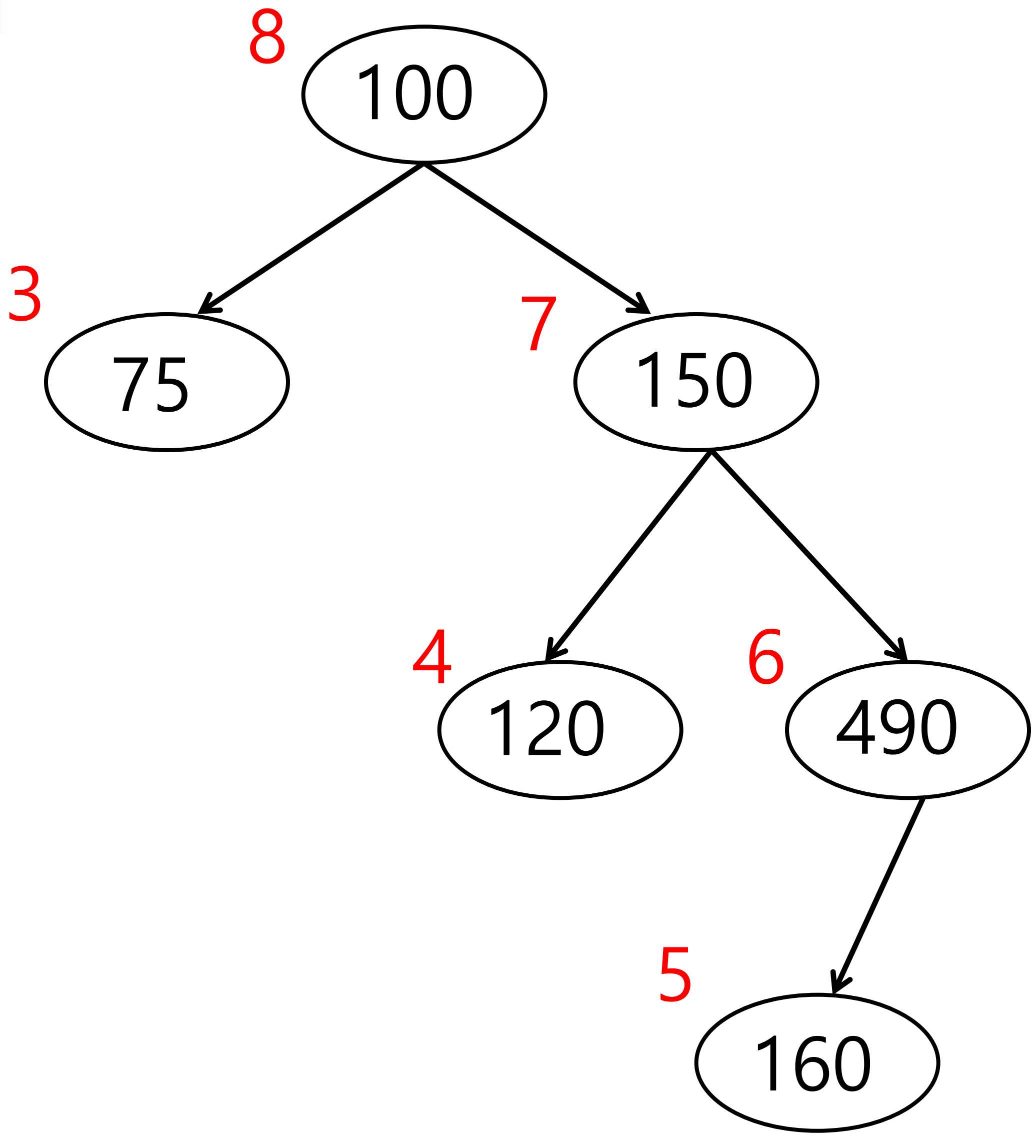
- First Child, Second Child, Mother
- Applications
  - (compiler) postfix expression evaluation
    - using a queue-like order, using a stack

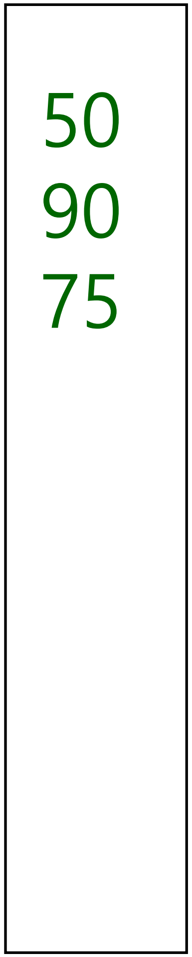
Postorder Traversal: Example (1/8)



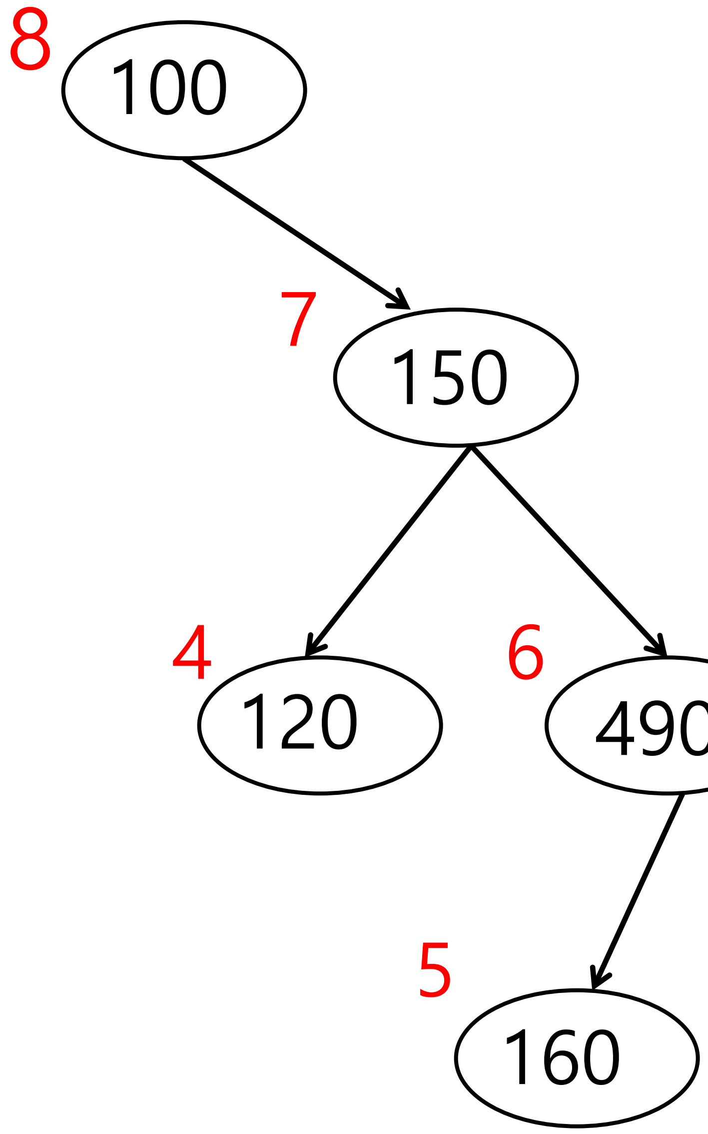


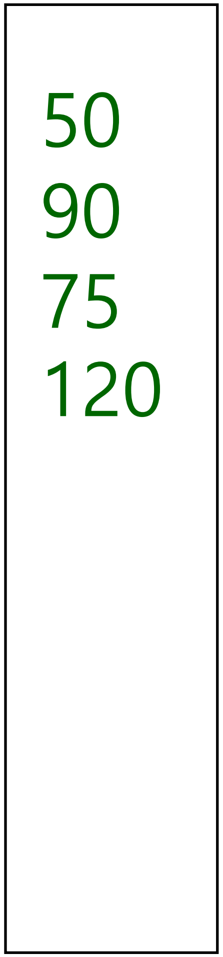
Postorder Traversal: Example (2/8)



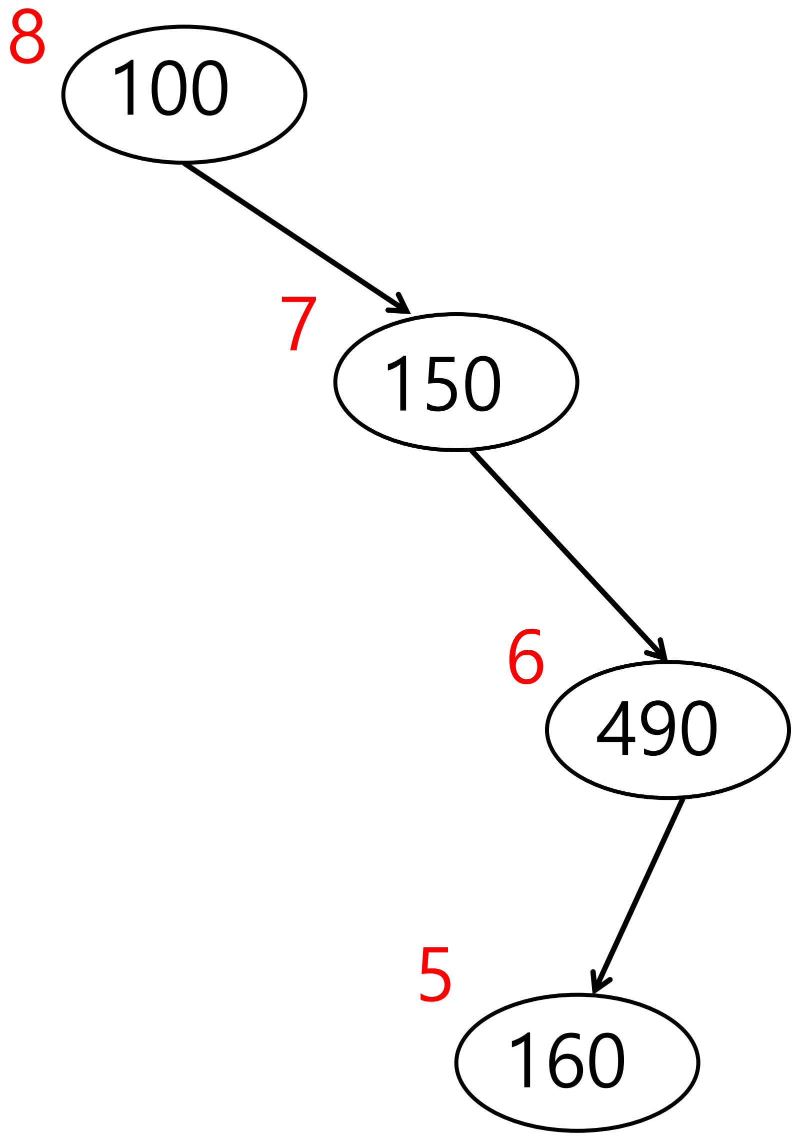


Postorder Traversal: Example (3/8)



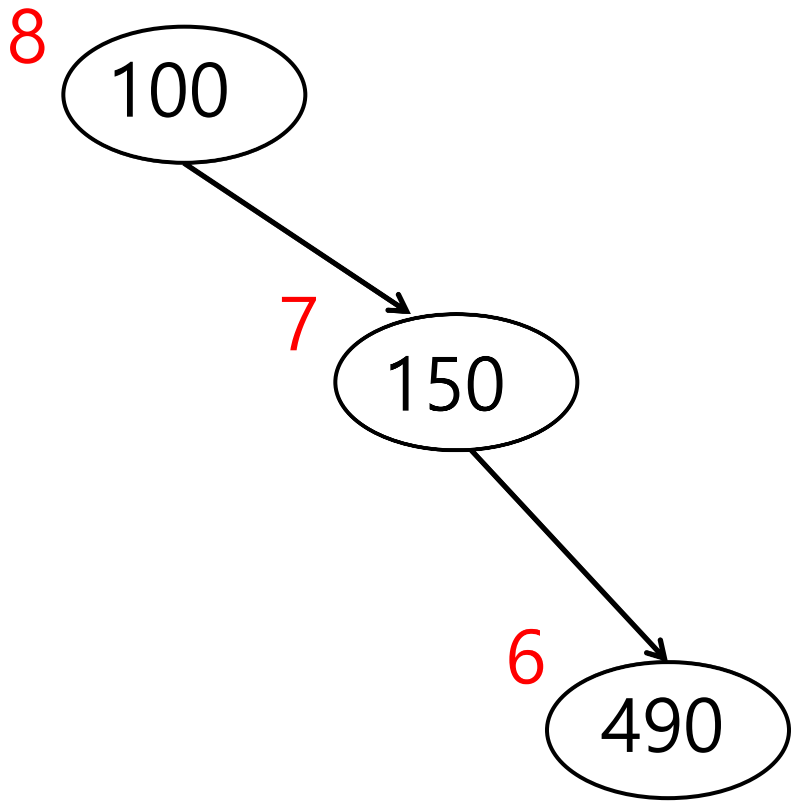


Postorder Traversal: Example (4/8)



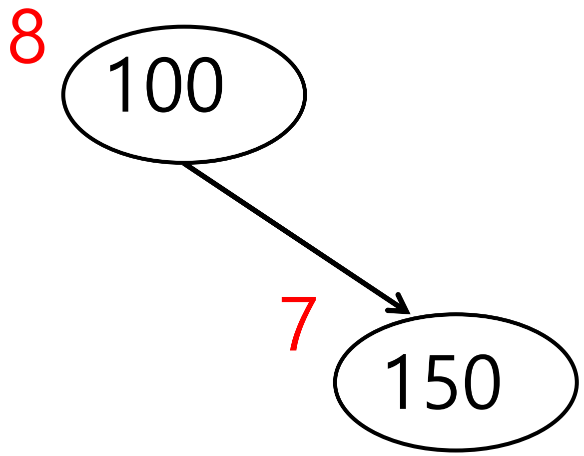
50
90
75
120
160

Postorder Traversal: Example (5/8)



50
90
75
120
160
490

Postorder Traversal: Example (6/8)



50
90
75
120
160
490
150

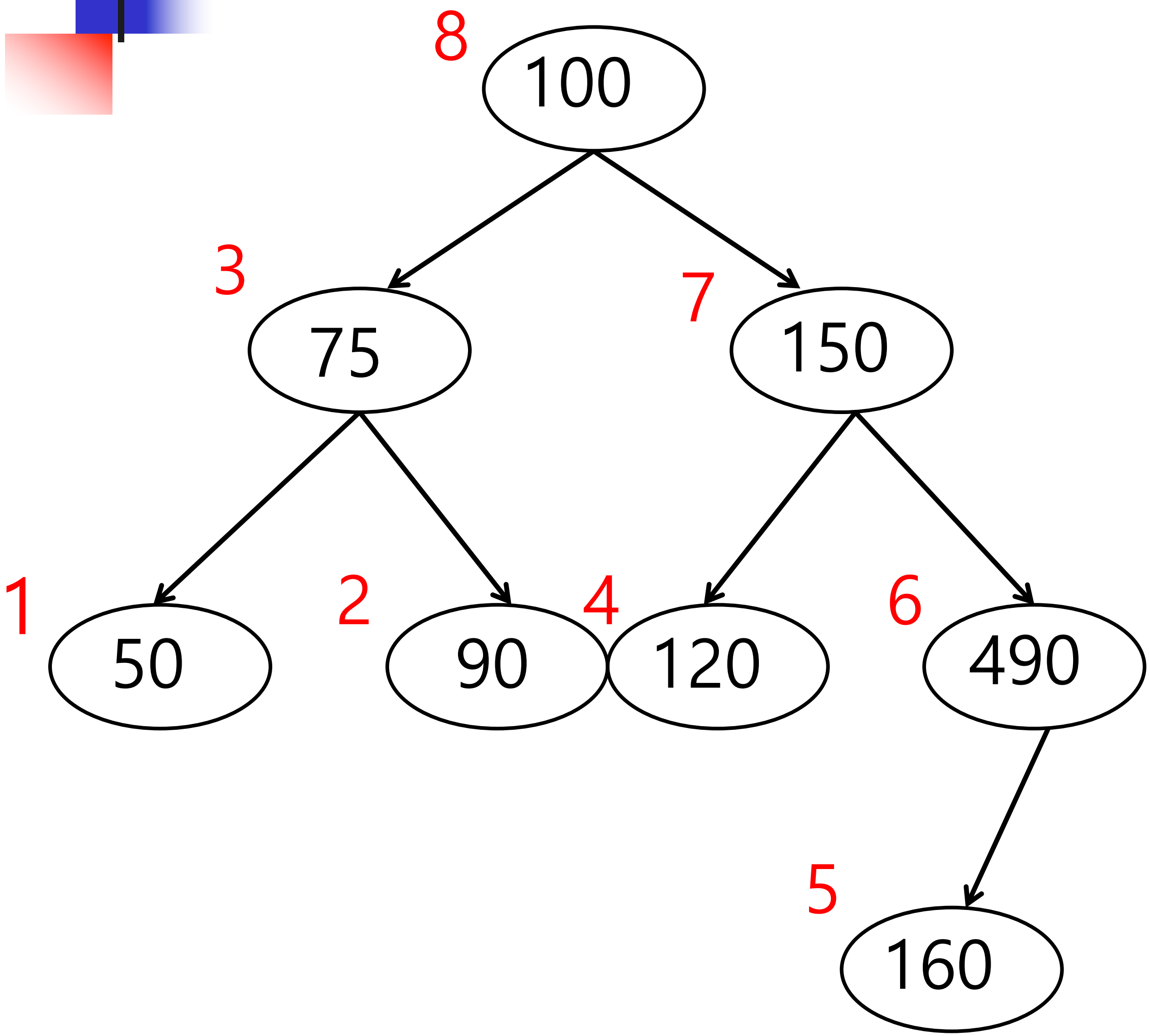
Postorder Traversal: Example (7/8)

8

100

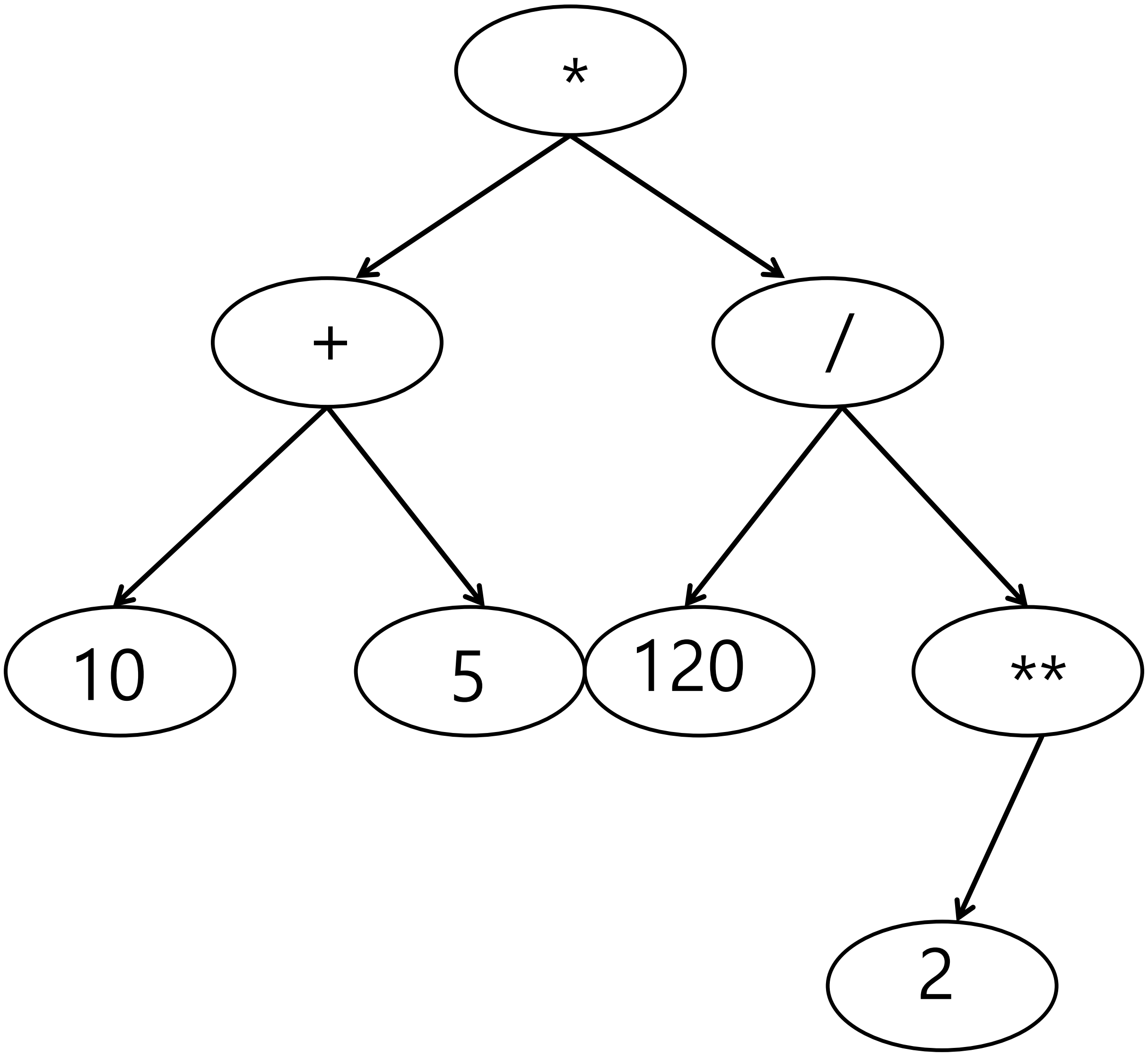
50
90
75
120
160
490
150
100

Postorder Traversal: Example (8/8)

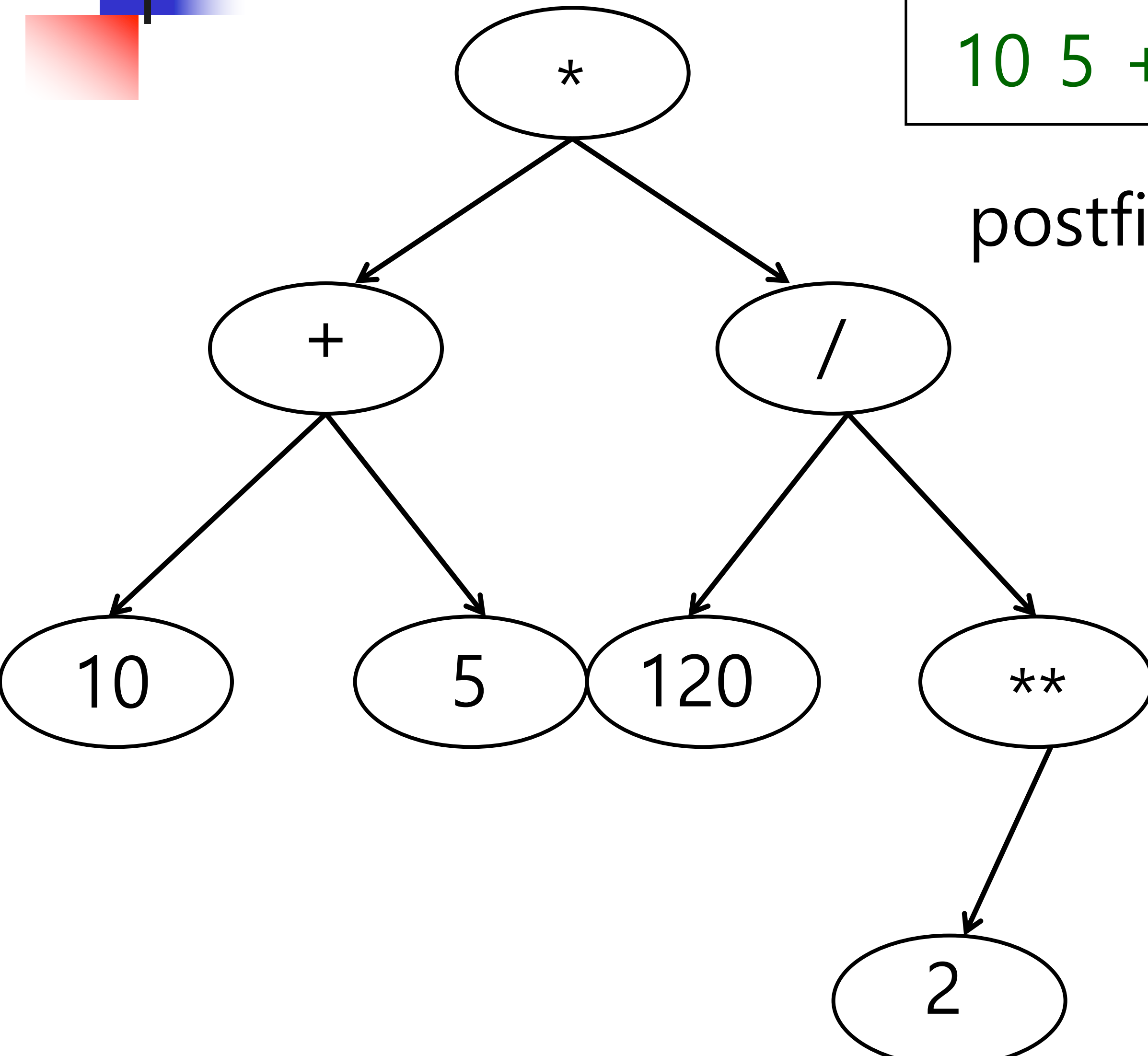


50
90
75
120
160
490
150
100

Postorder Traversal: Exercise



Postorder Traversal: Solution



10 5 + 120 2 ** / *

postfix expression

Evaluating a Postfix Expression Using a Stack (How compilers evaluate expressions) (1/5)

Pop stack-1.
If it is a number,
    push it to stack-2
If it is an operator,
    pop stack-2 and compute,
Push the result to stack-2

reverse stack

10 5 + 120 2 ** / *

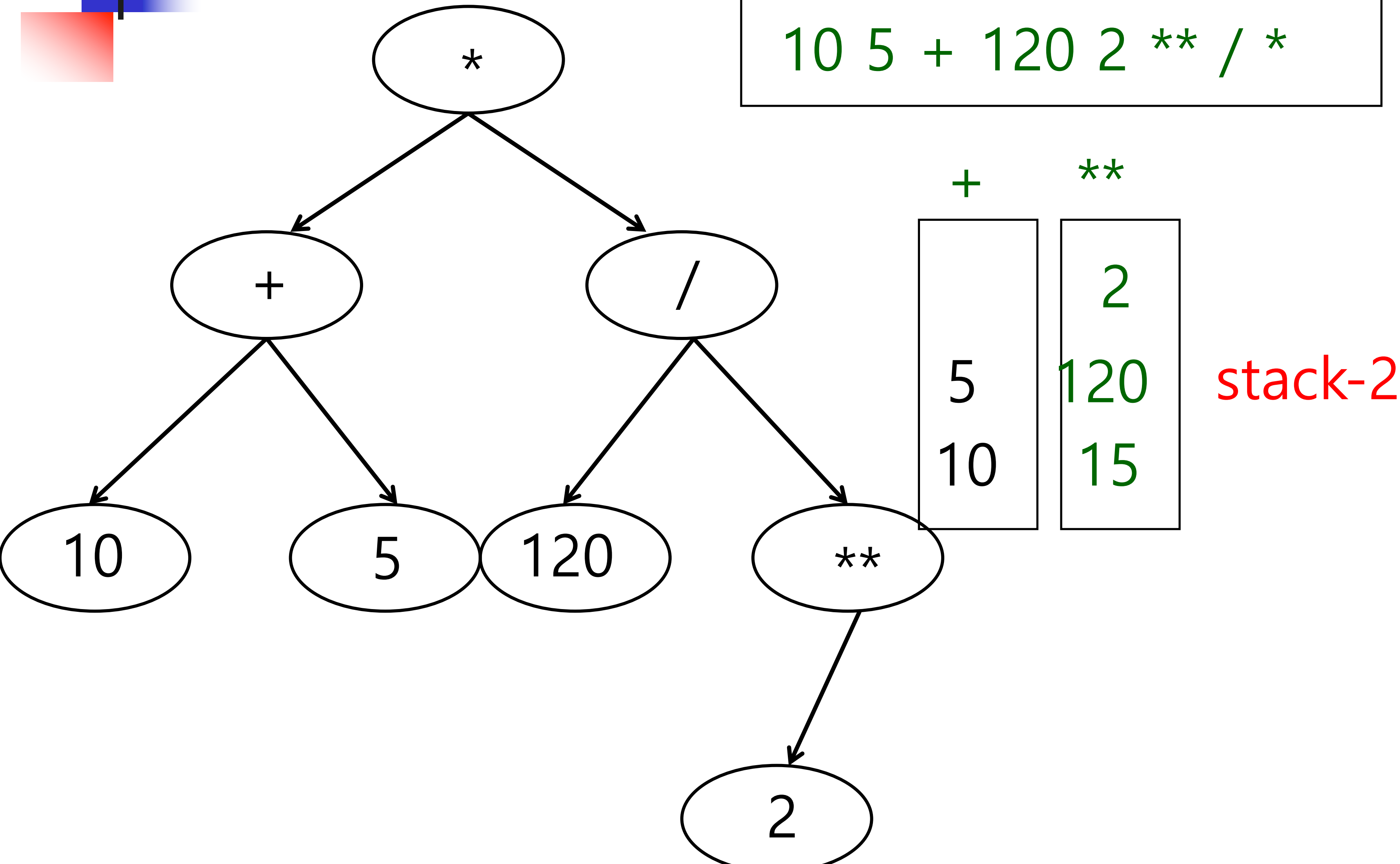
+

5
10

stack-2

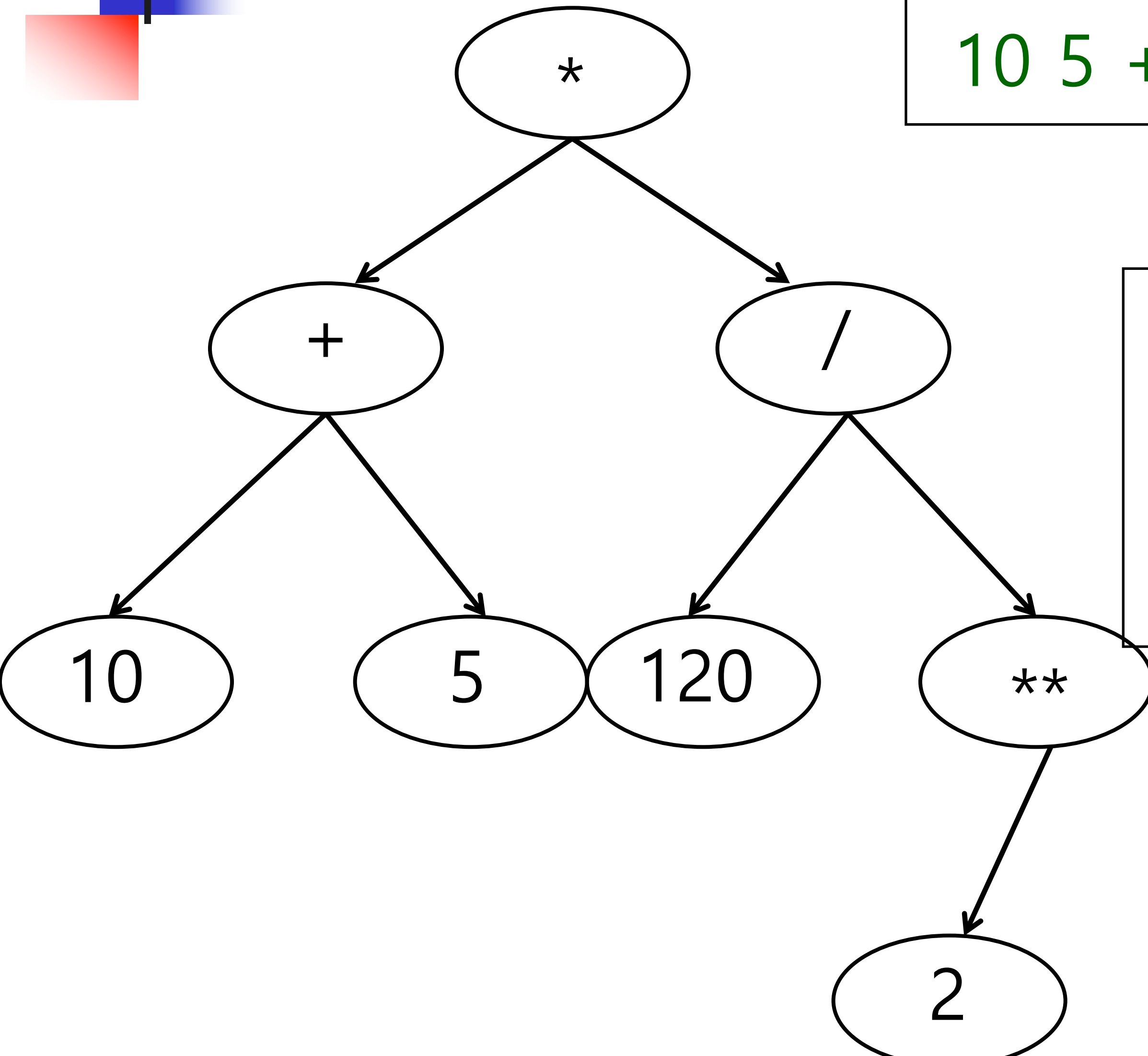
(2/5)

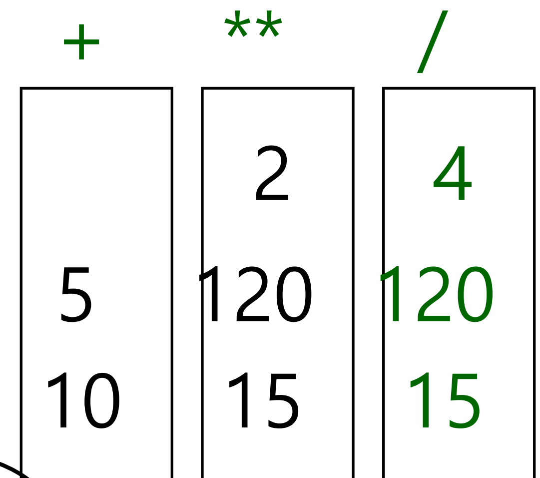
10 5 + 120 2 ** / *



(3/5)

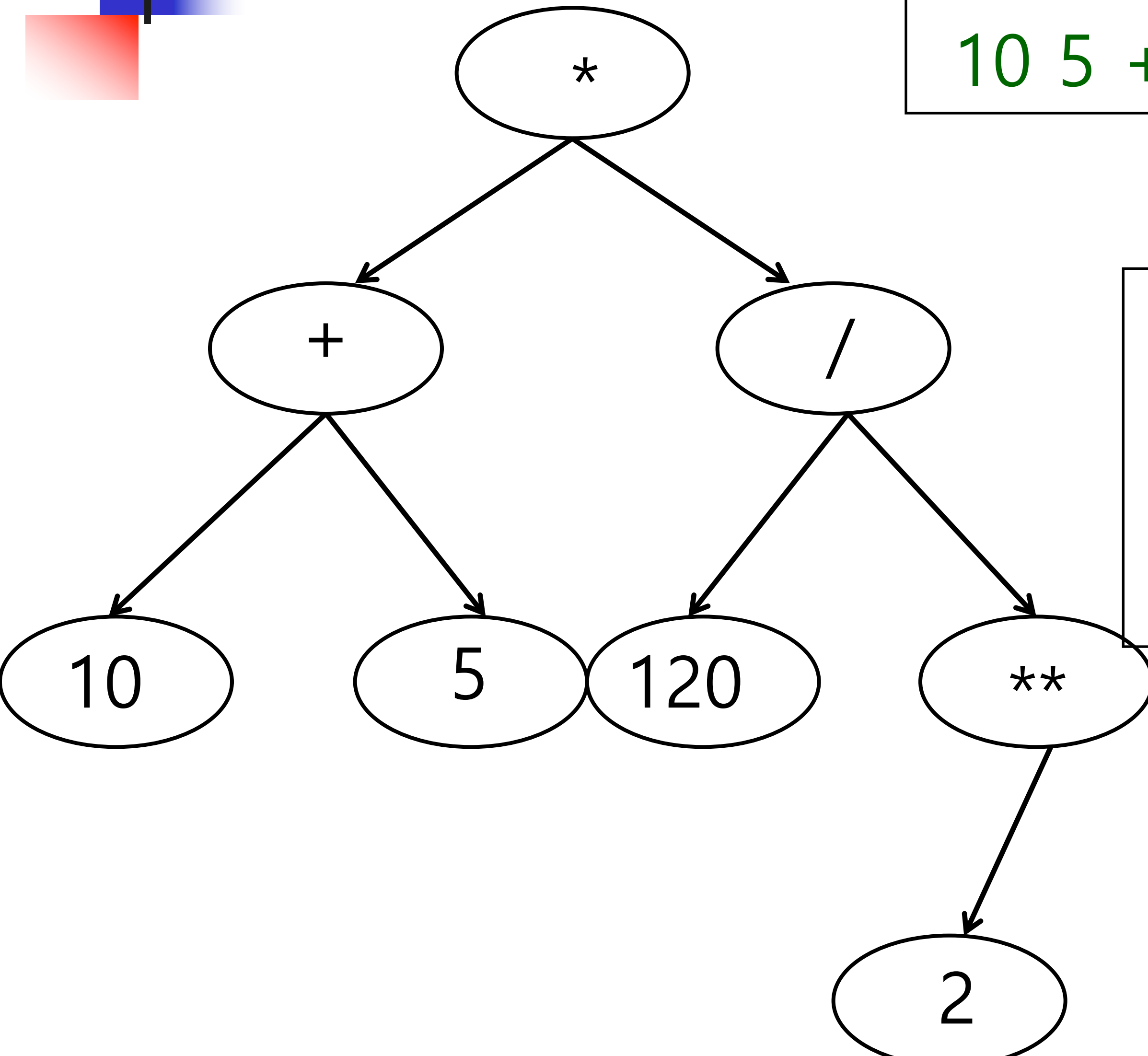
10 5 + 120 2 ** / *

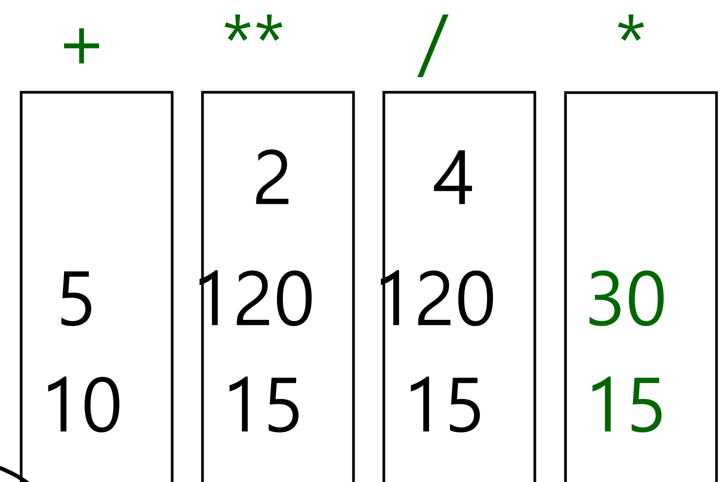




(4/5)

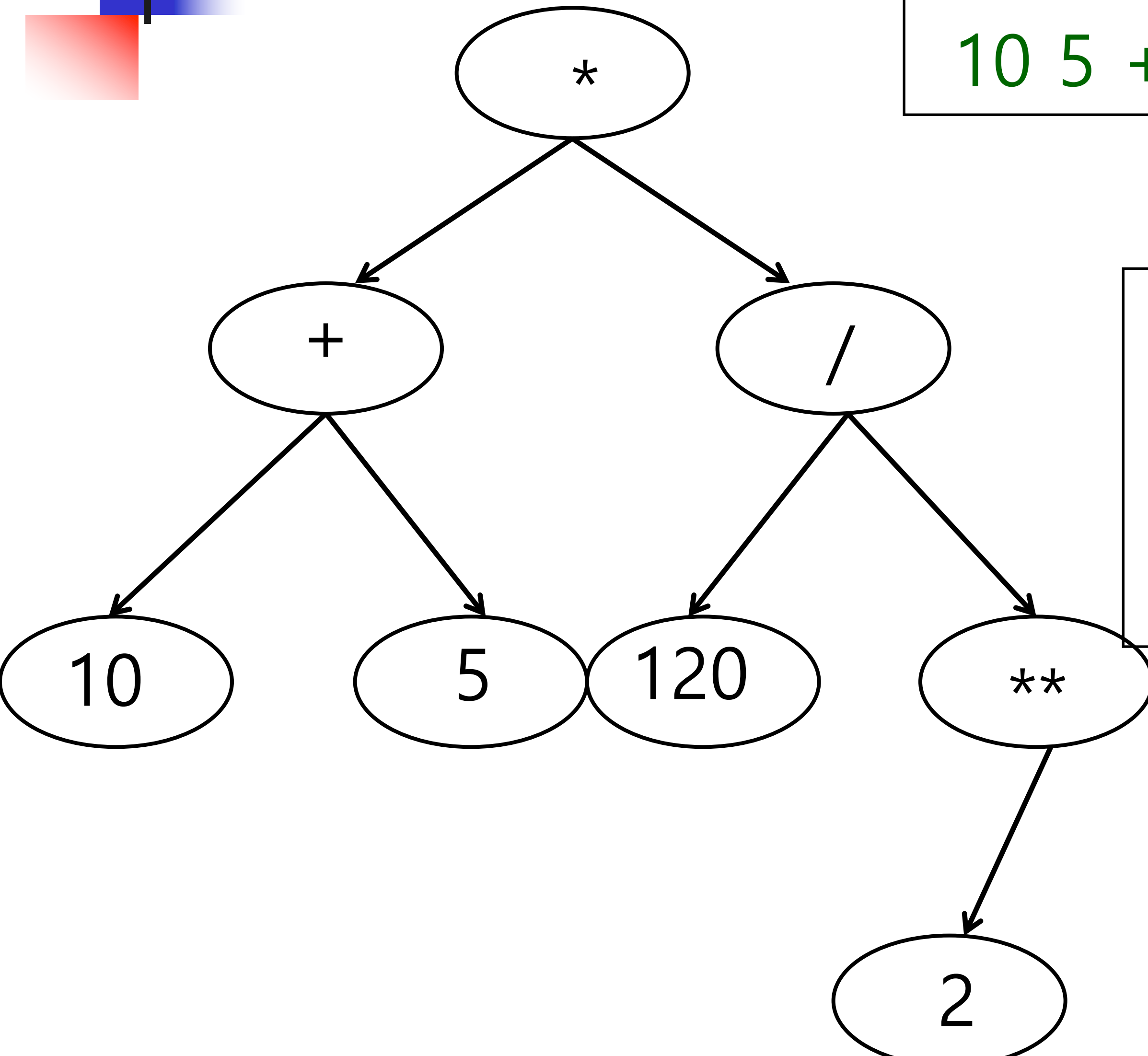
10 5 + 120 2 ** / *

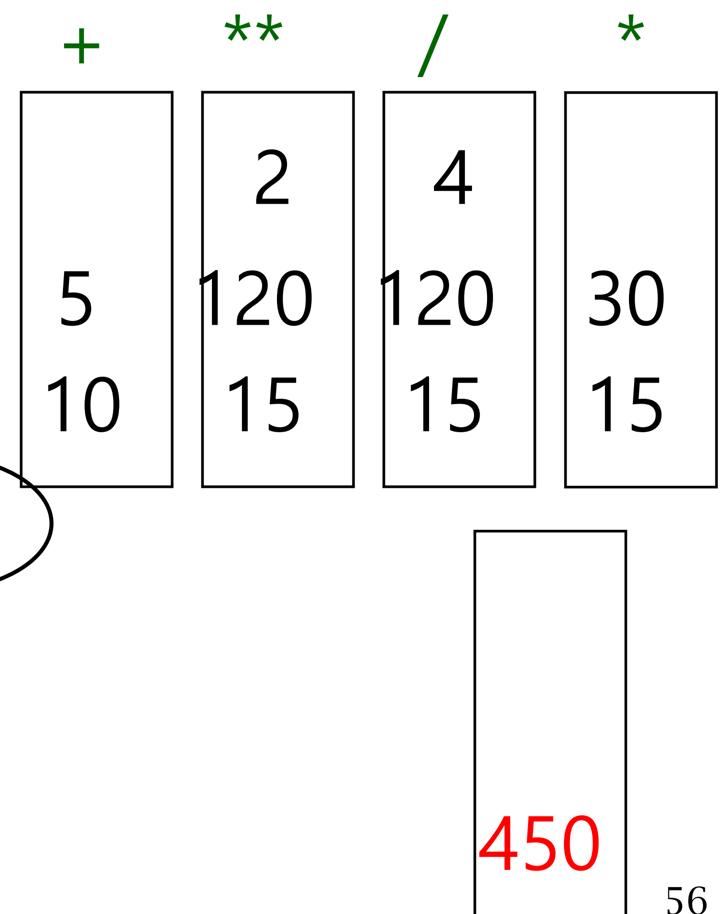




(5/5)

10 5 + 120 2 ** / *

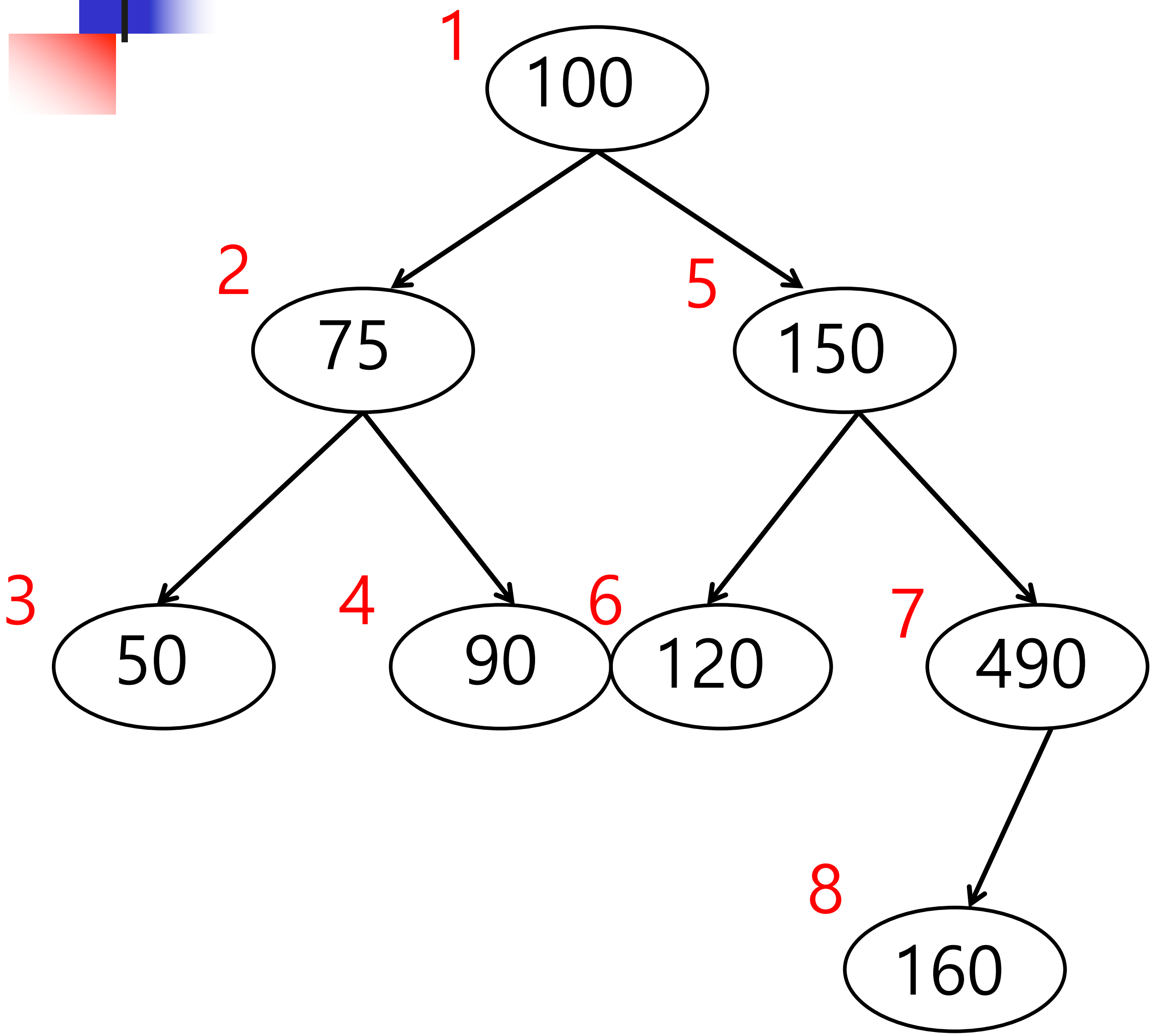




Preorder Traversal

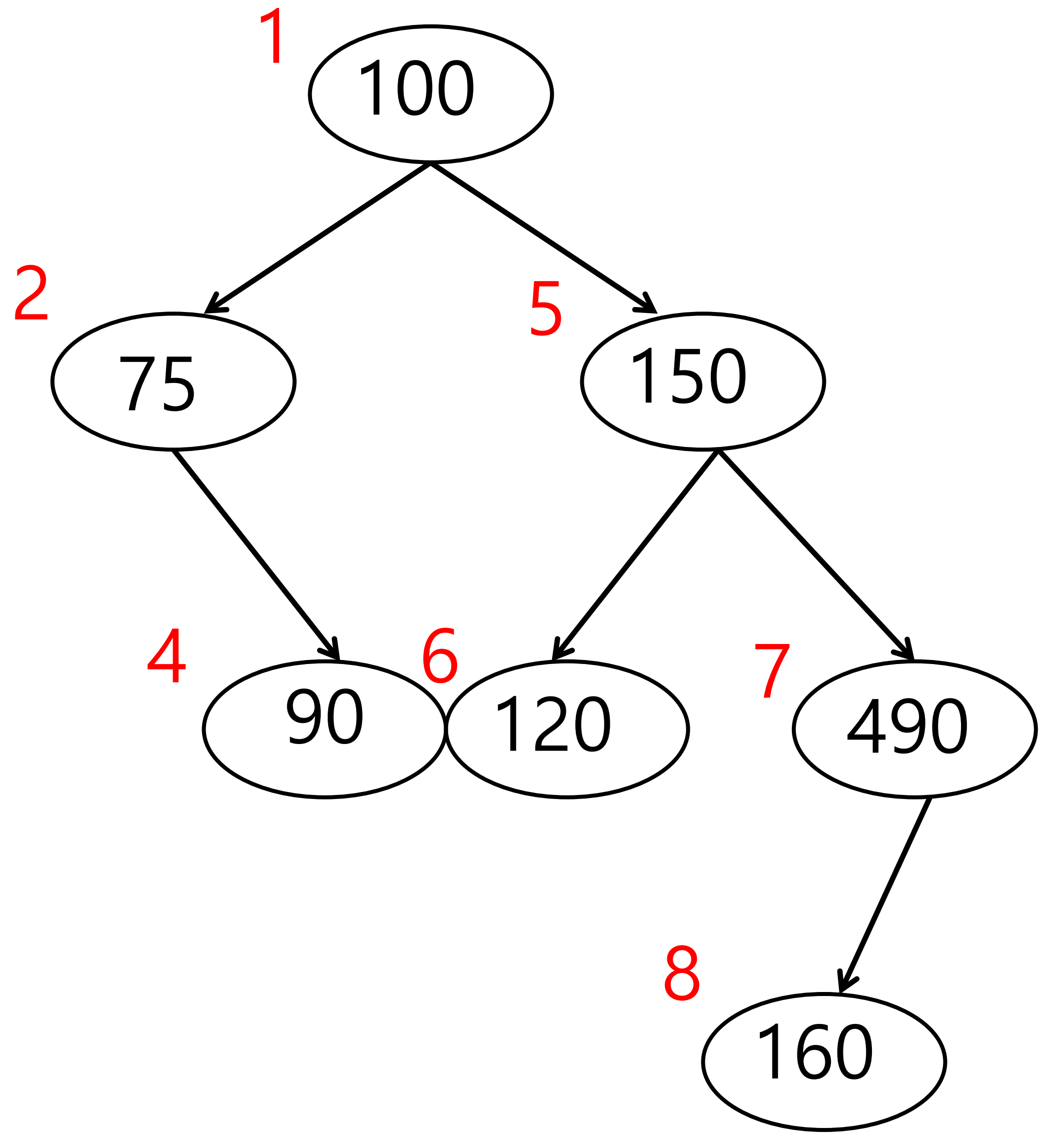
- Mother, First Child, Second Child
- Applications
  - Make a complete copy of a tree
  - (compiler) prefix expression evaluation
    - In stack-like (reverse) order, using a stack

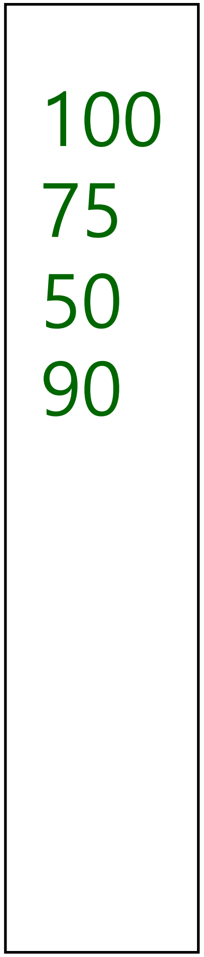
Preorder Traversal: Example (1/6)



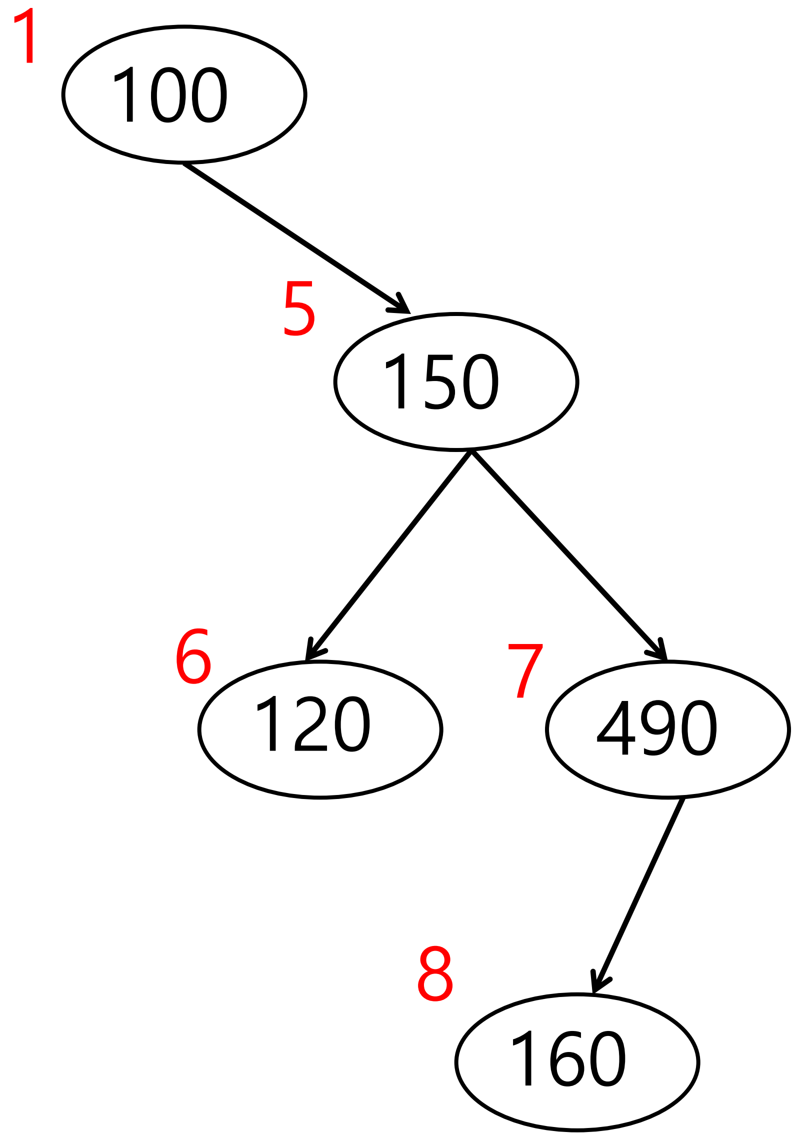
100
75
50

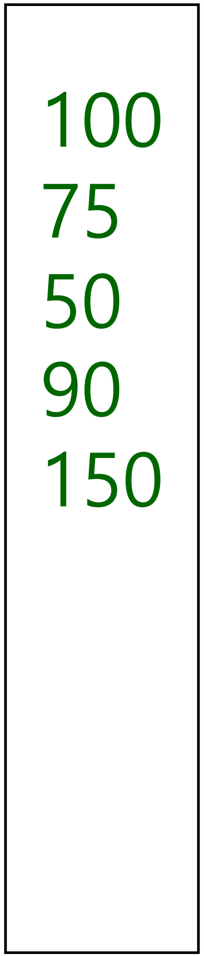
Preorder Traversal: Example (2/6)



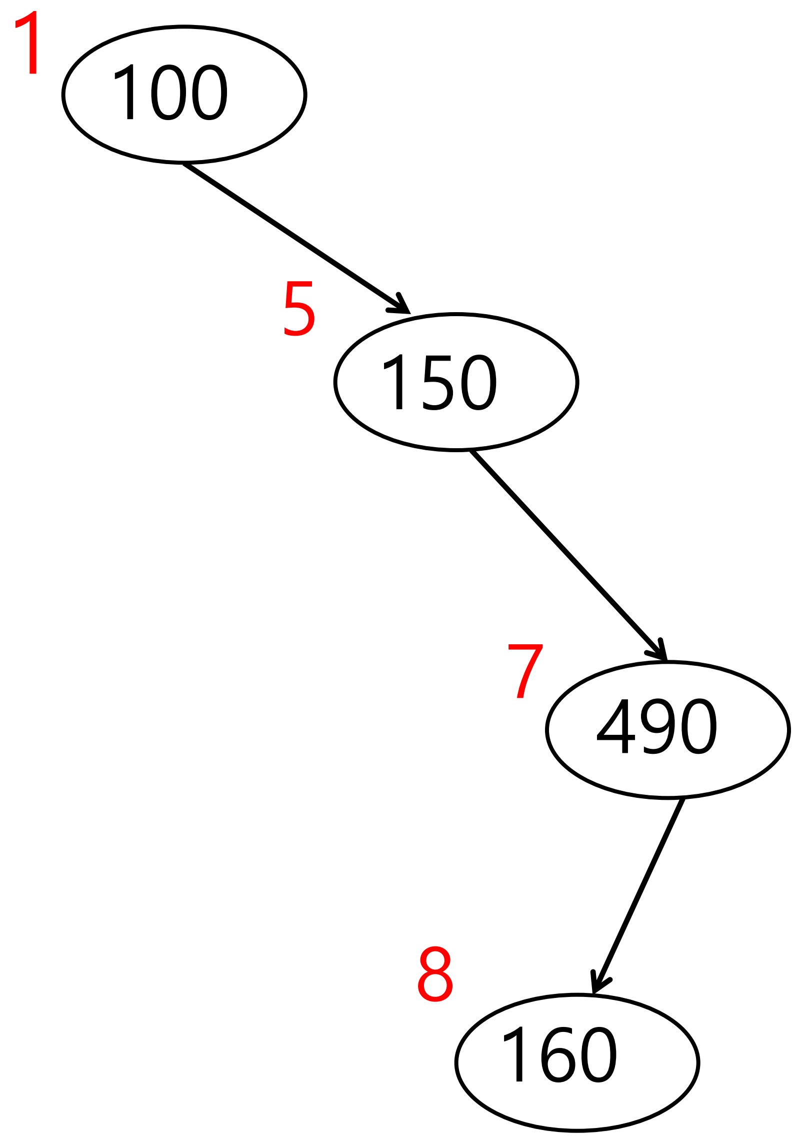


Preorder Traversal: Example (3/6)



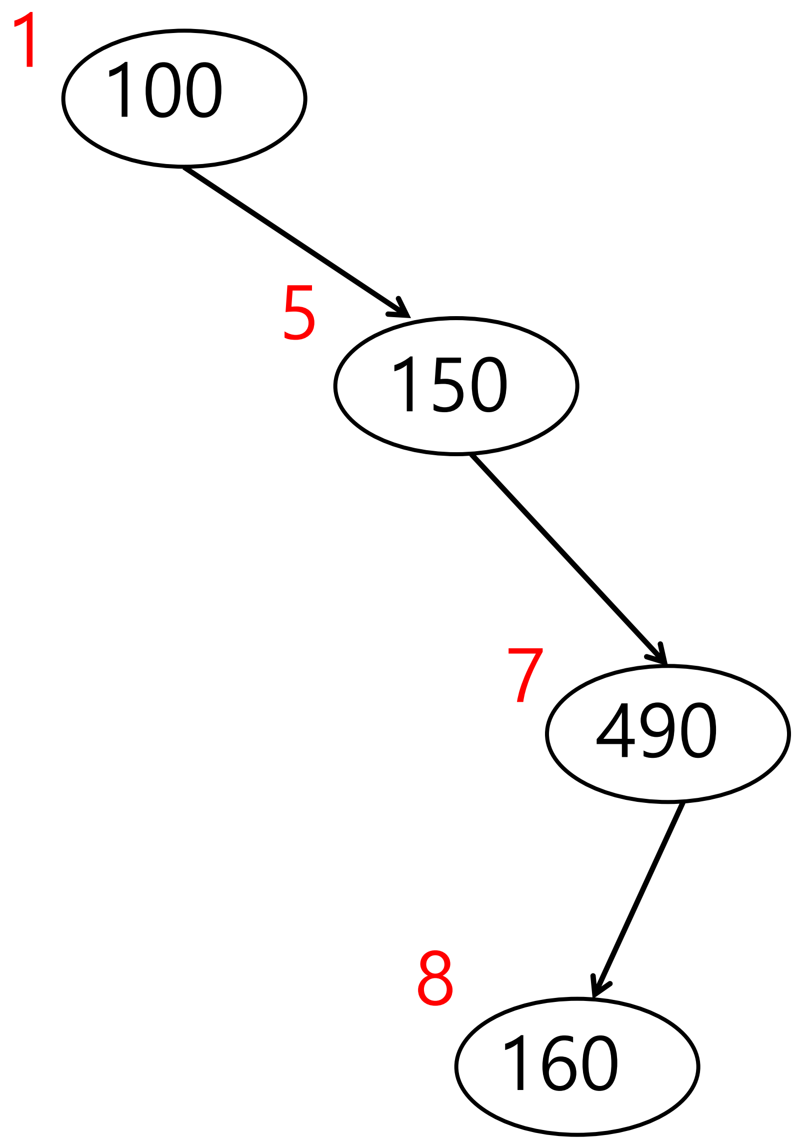


Preorder Traversal: Example (4/6)



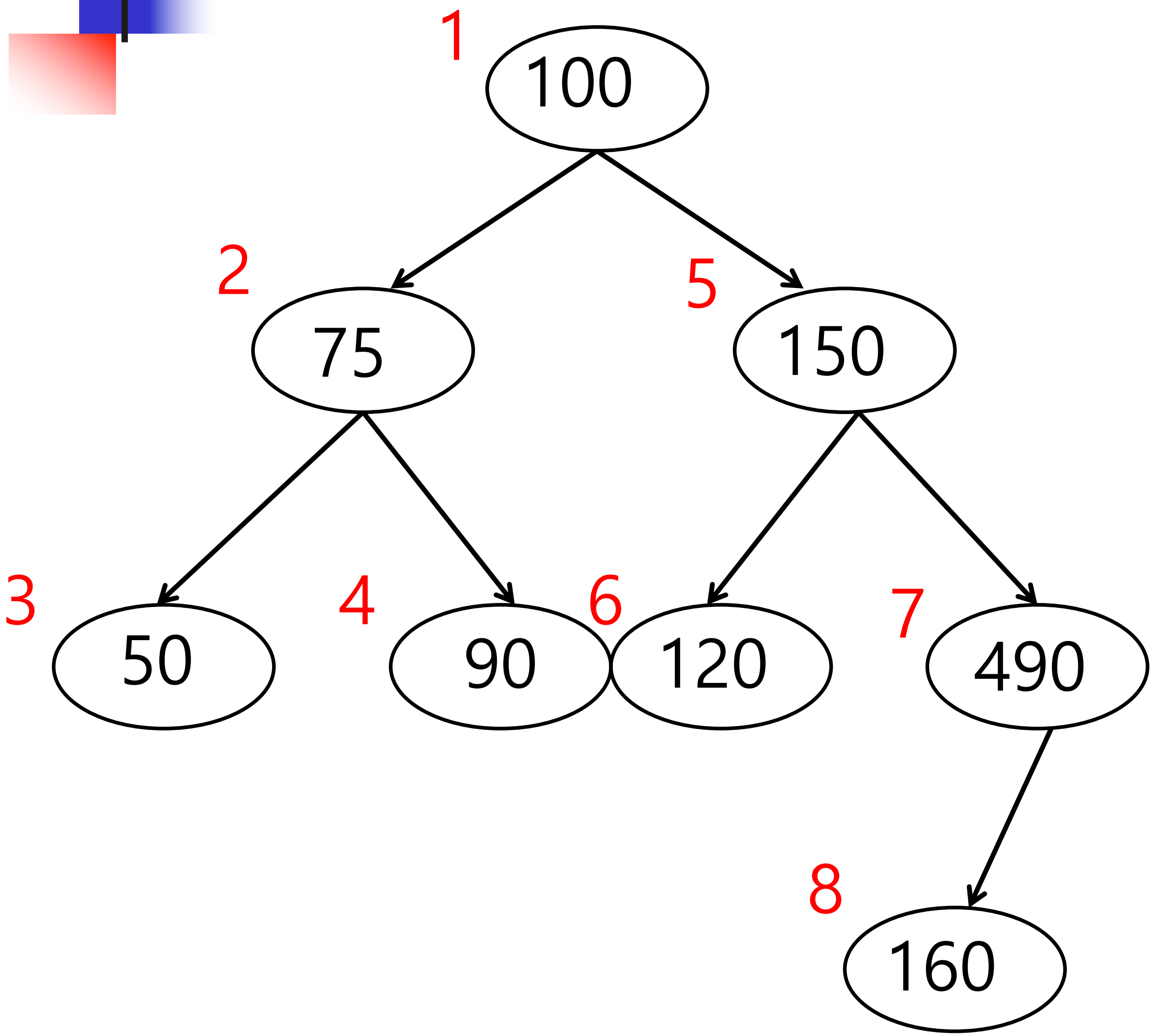
100
75
50
90
150
120

Preorder Traversal: Example (5/6)



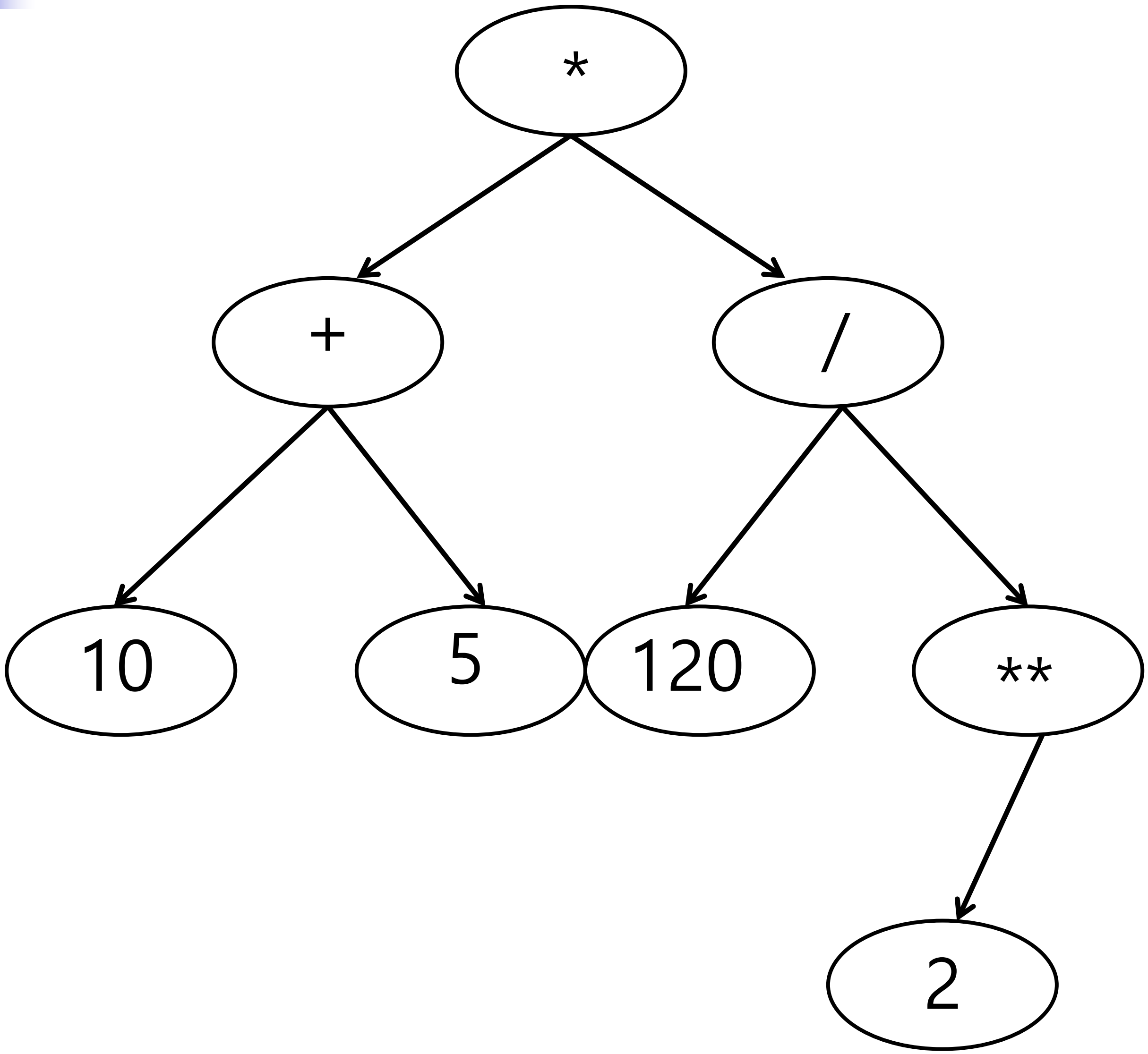
100
75
50
90
150
120
490
160

Preorder Traversal: Example (6/6)

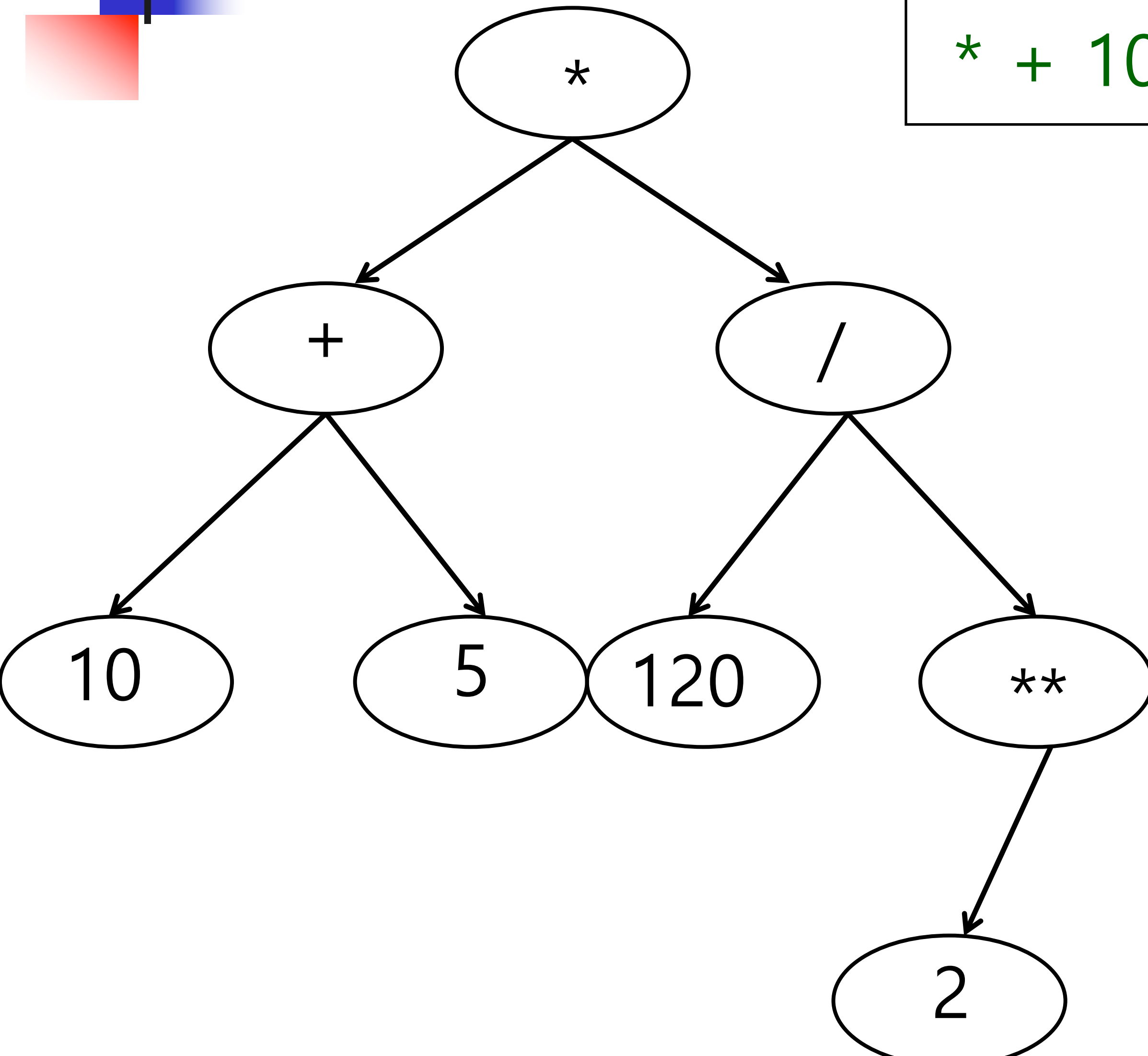


100
75
50
90
150
120
490
160

Preorder Traversal: Exercise

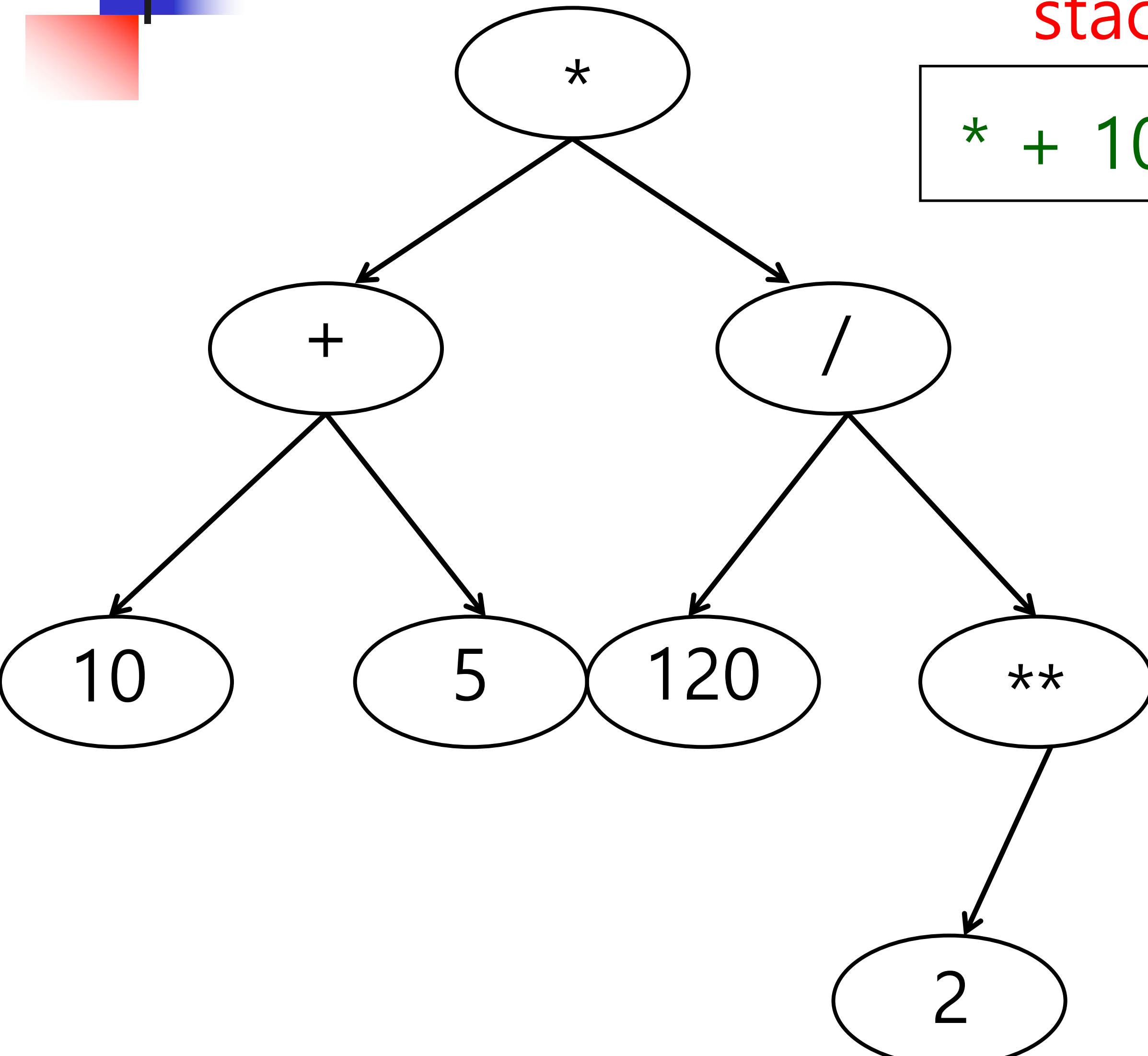


Preorder Traversal: Solution



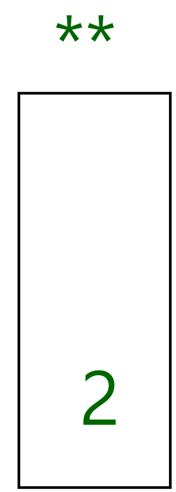
* + 10 5 / 120 ** 2

Evaluating a Prefix Expression Using a Stack (1/5)

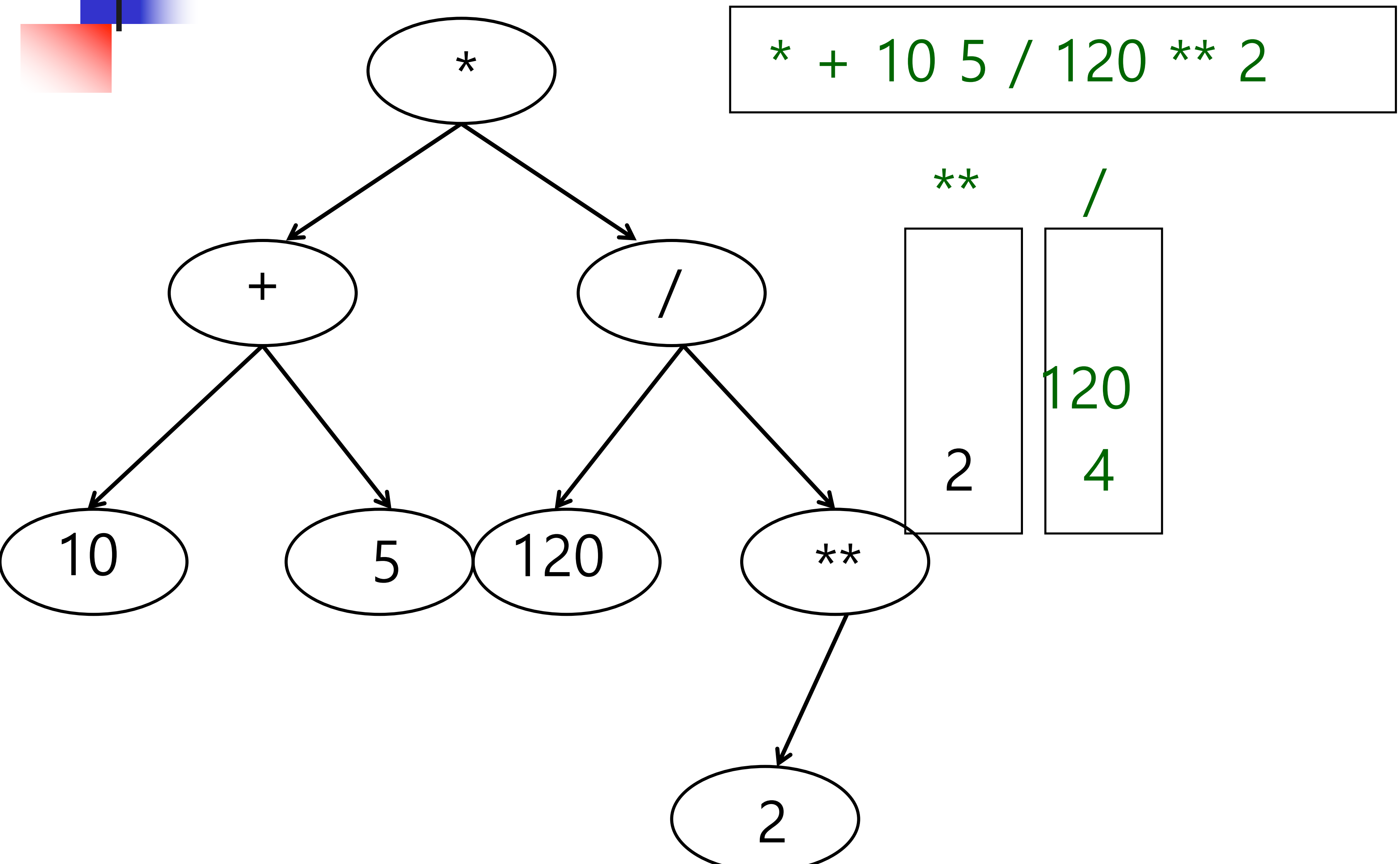


stack-1

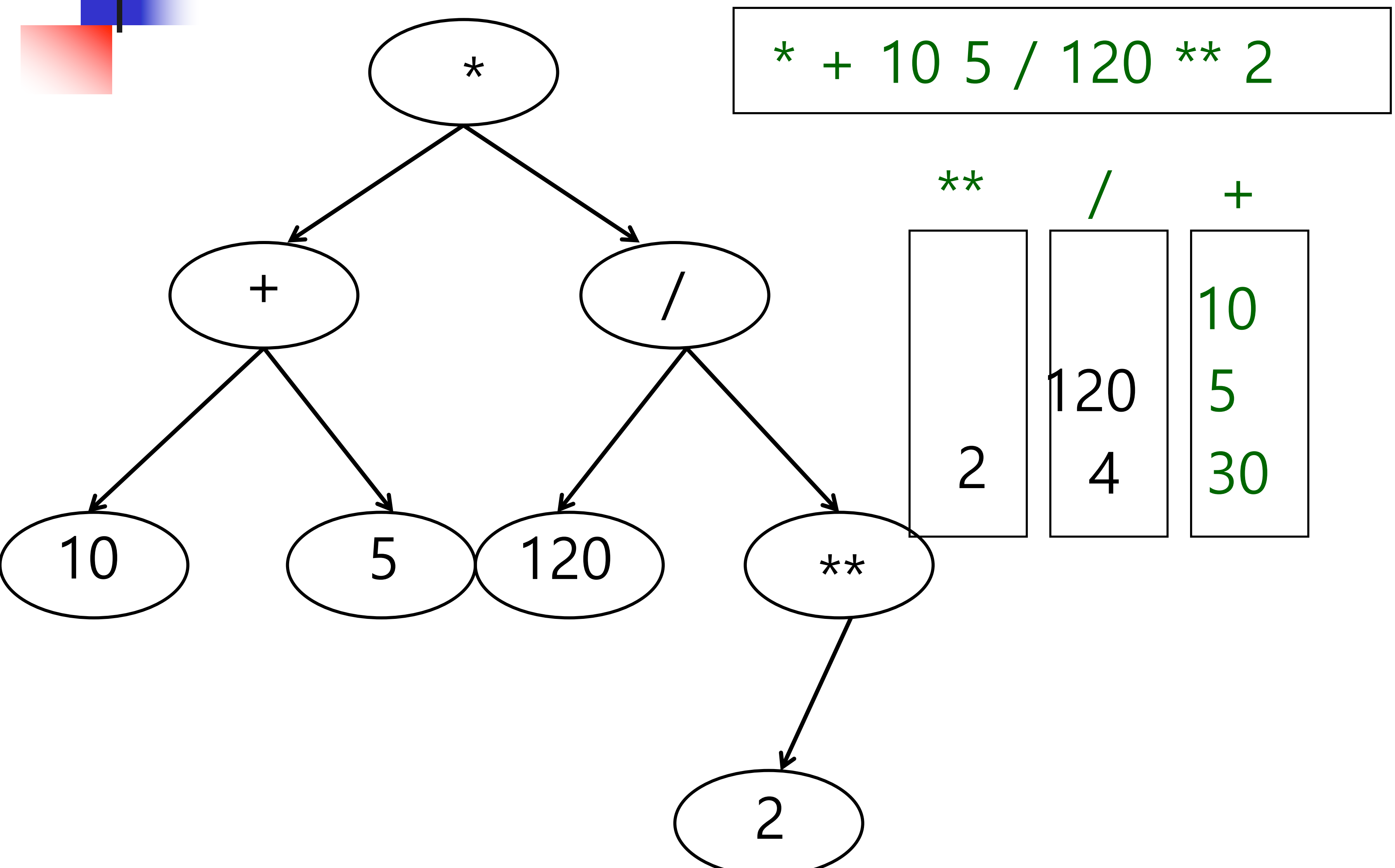
* + 10 5 / 120 ** 2



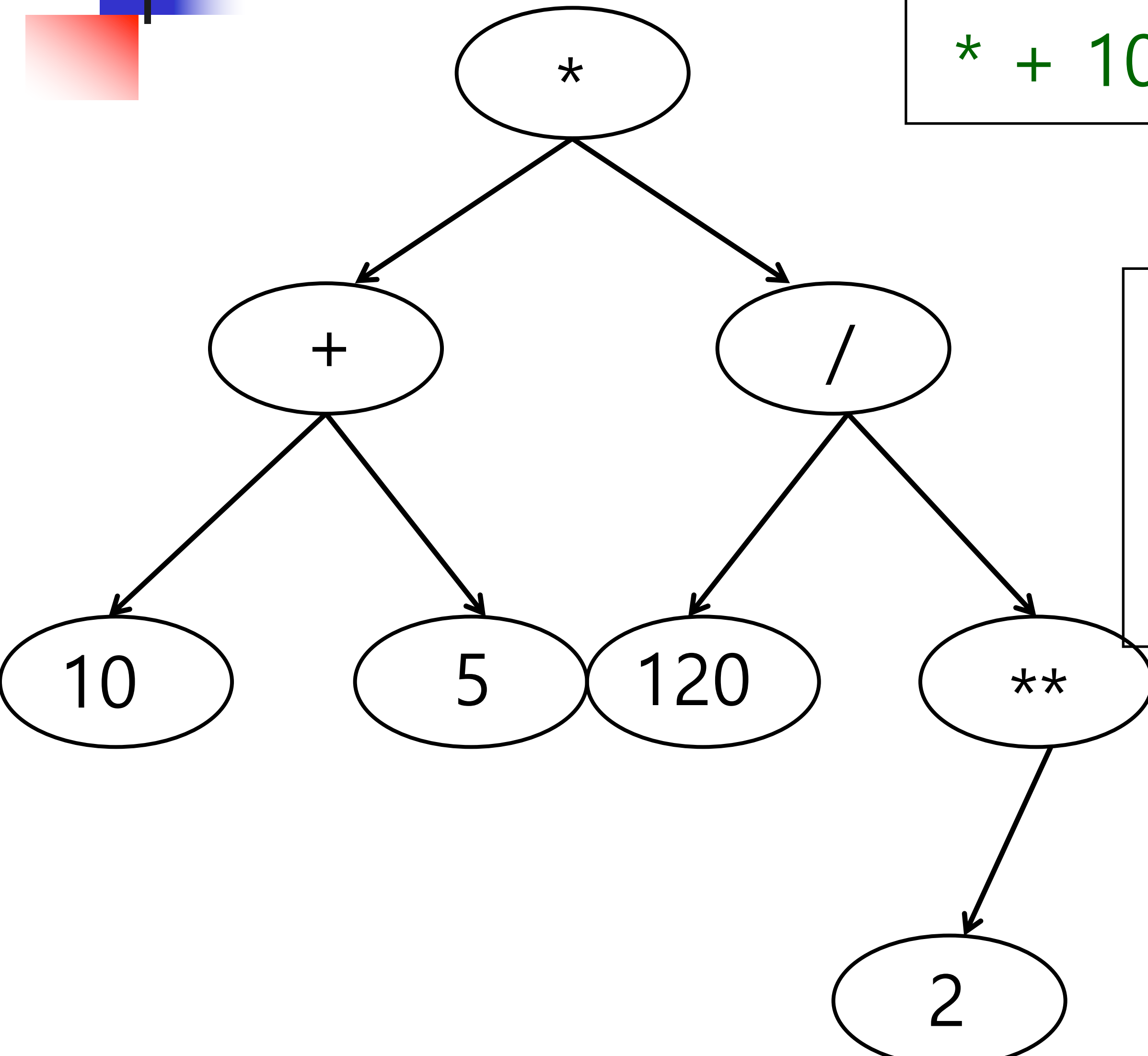
(2/5)



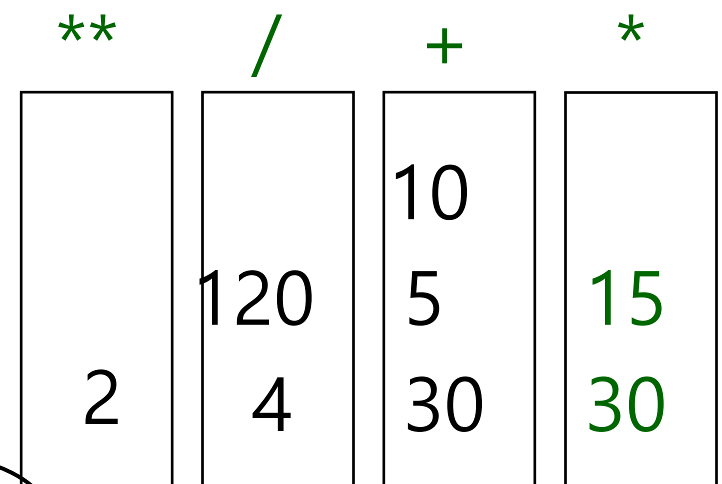
(3/5)



(4/5)

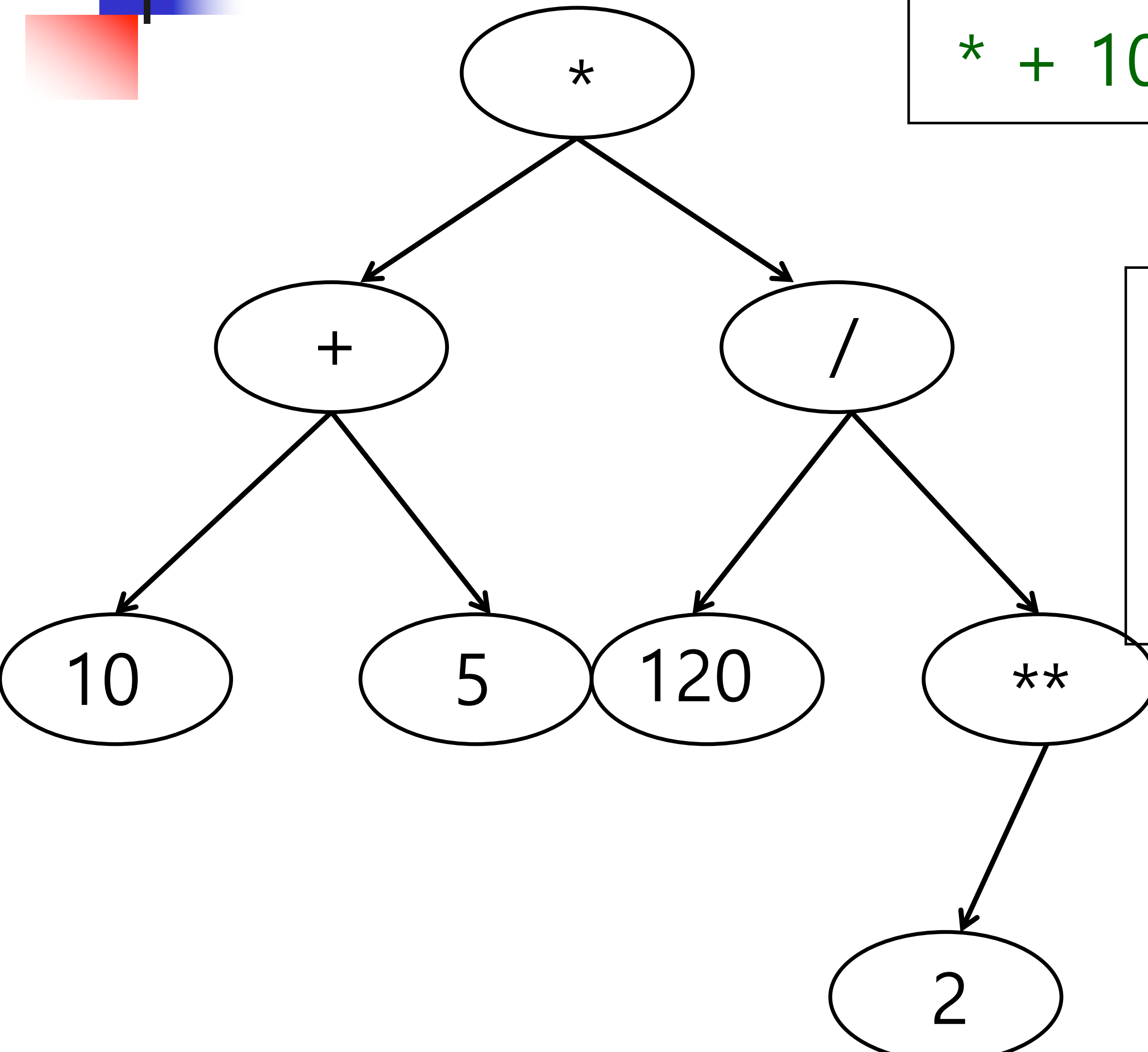


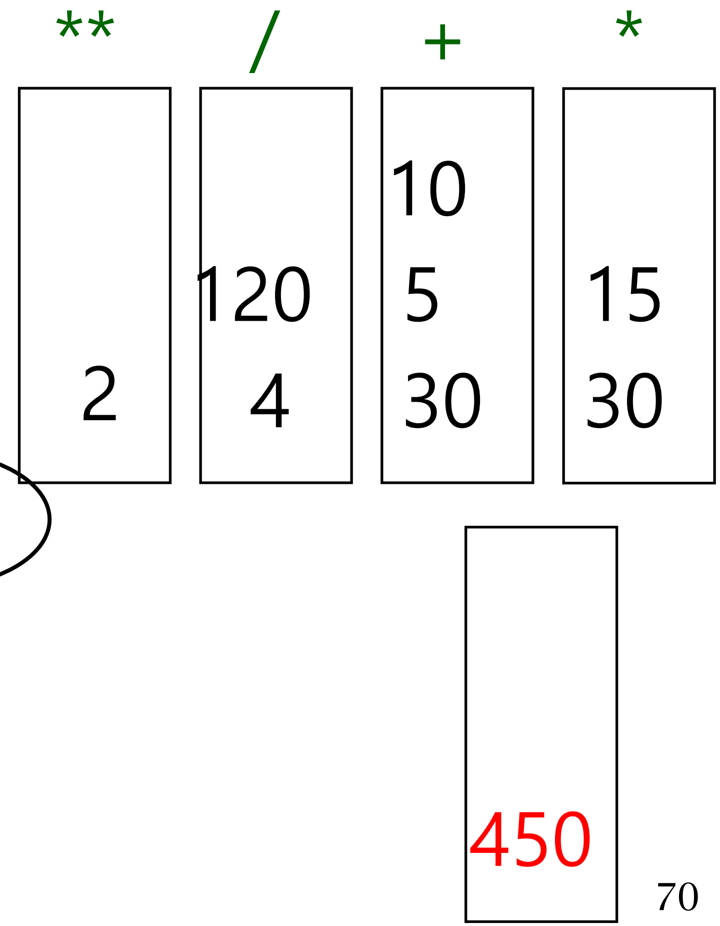
* + 10 5 / 120 ** 2



(5/5)

* + 10 5 / 120 ** 2





Level Order Traversal

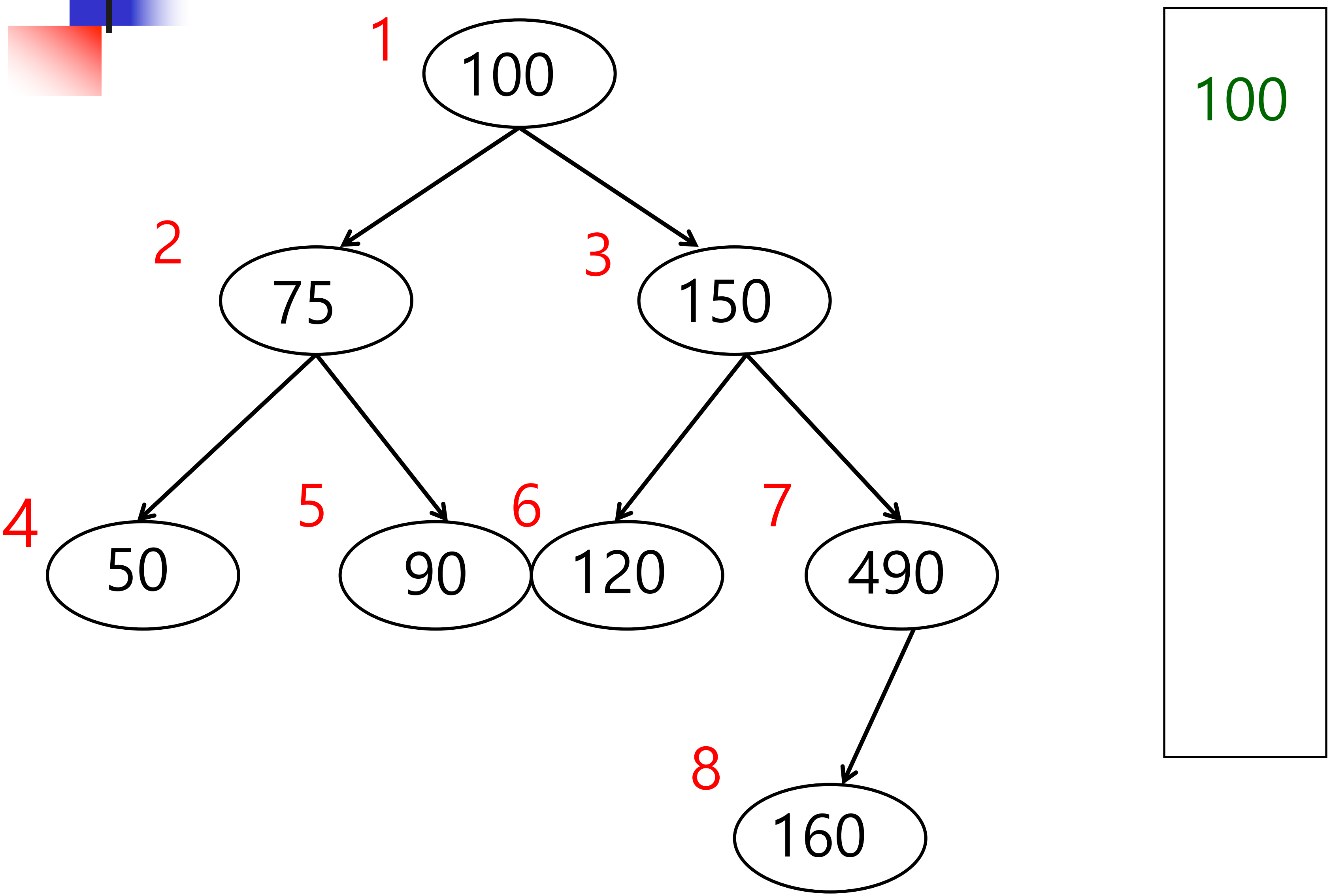
- Range (Level) Retrieval
  - People of the same rank on an organizational chart
  - Groups of the same rank on an organization chart
  - Possible next moves in a chess/go game

Playing Chess

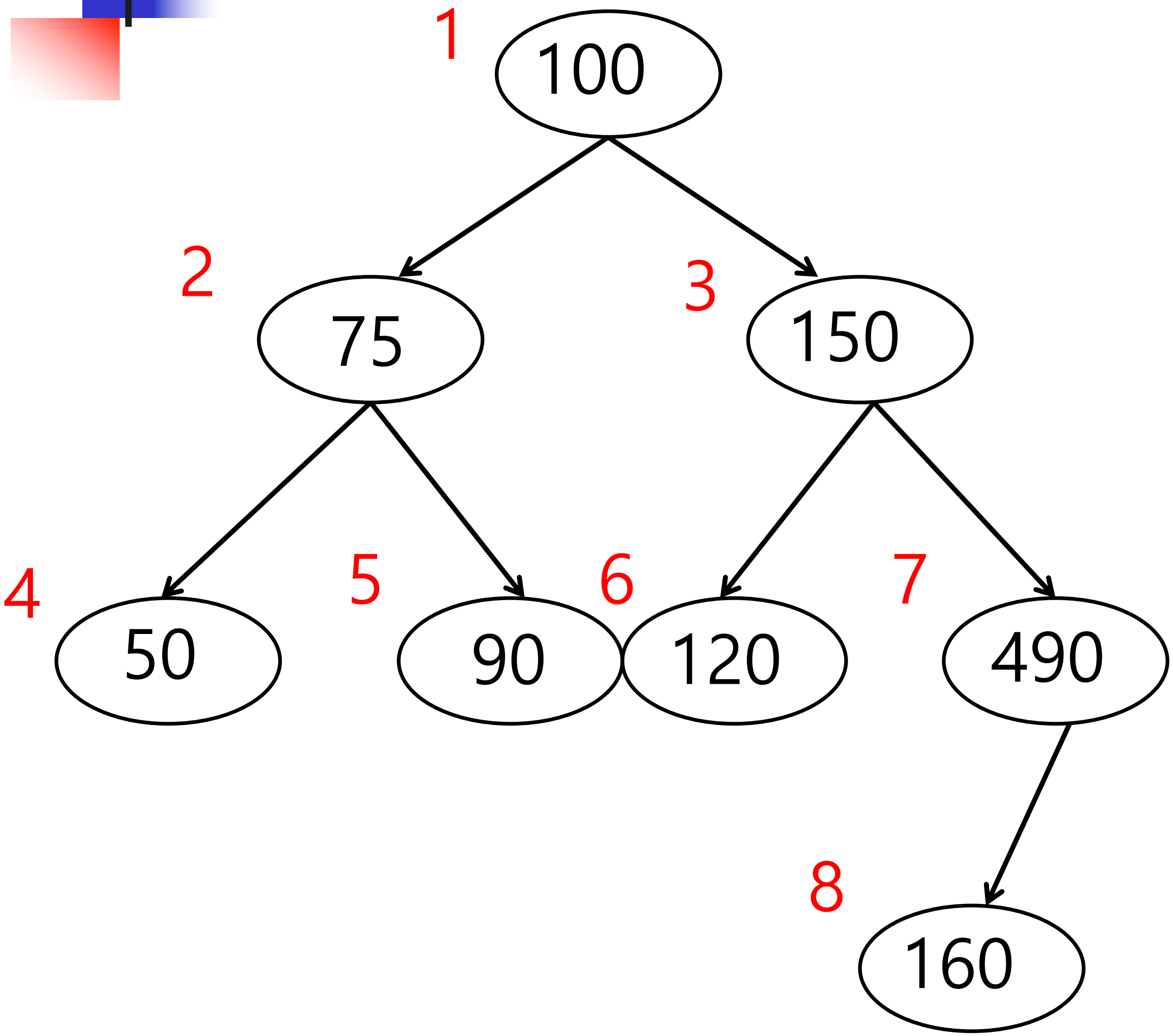
IBM Deep Blue – Chess Playing Computer

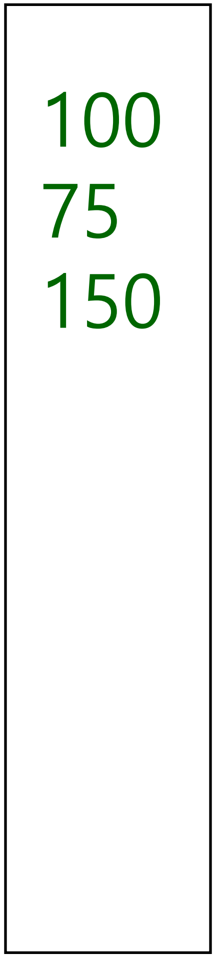
Playing Go

Level Order Traversal: Example (1/4)

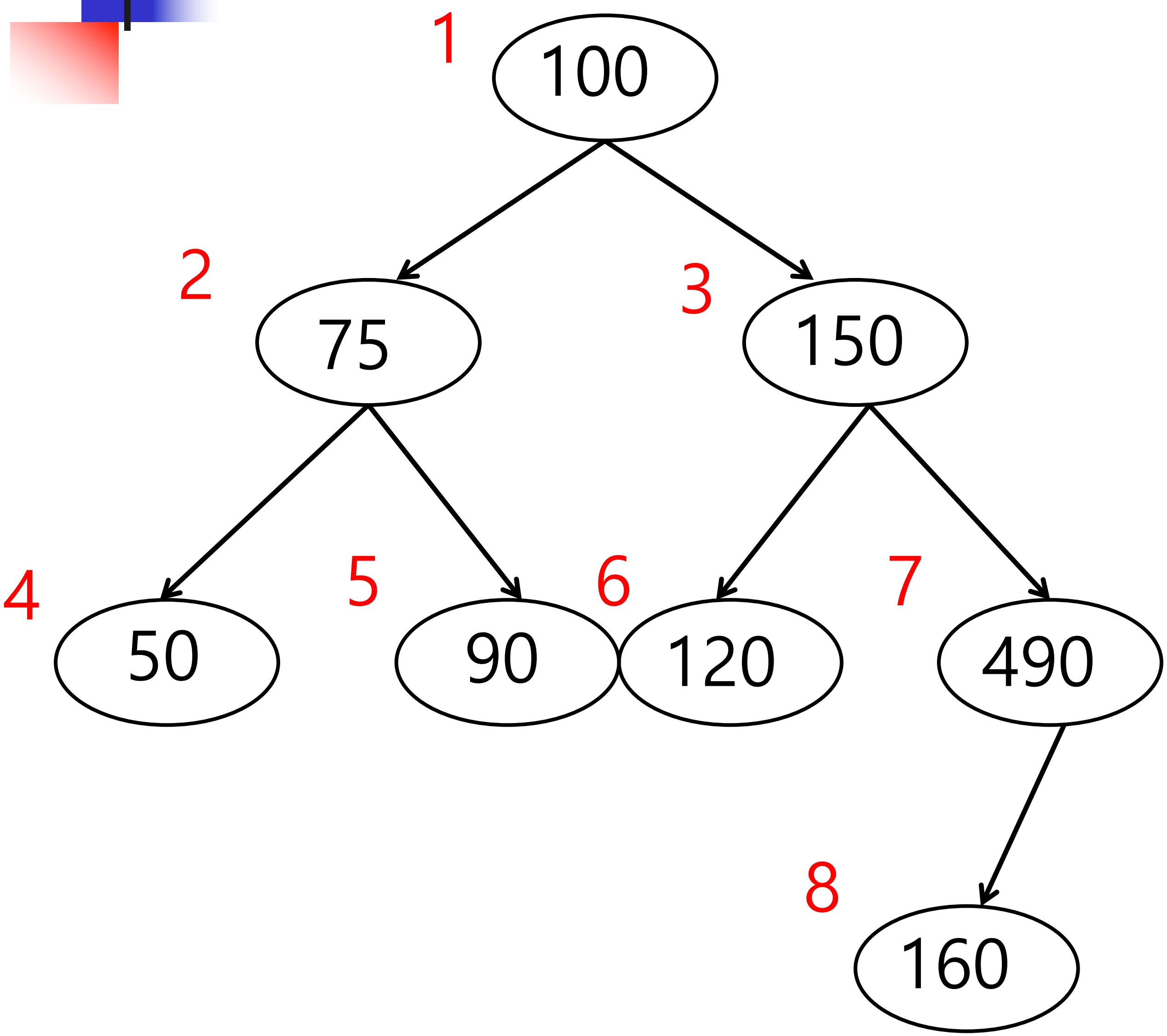


Level Order Traversal: Example (2/4)



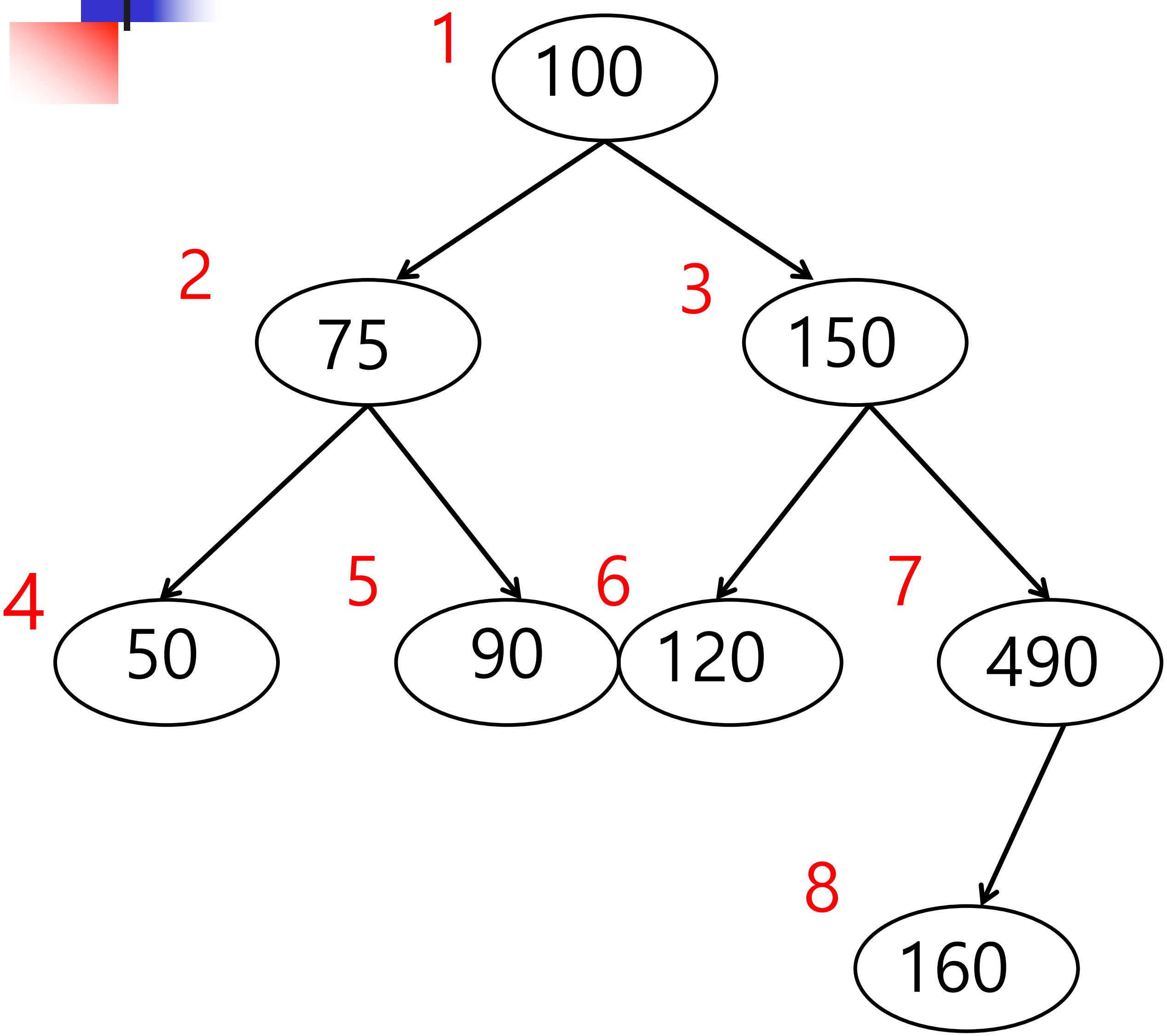


Level Order Traversal: Example (3/4)



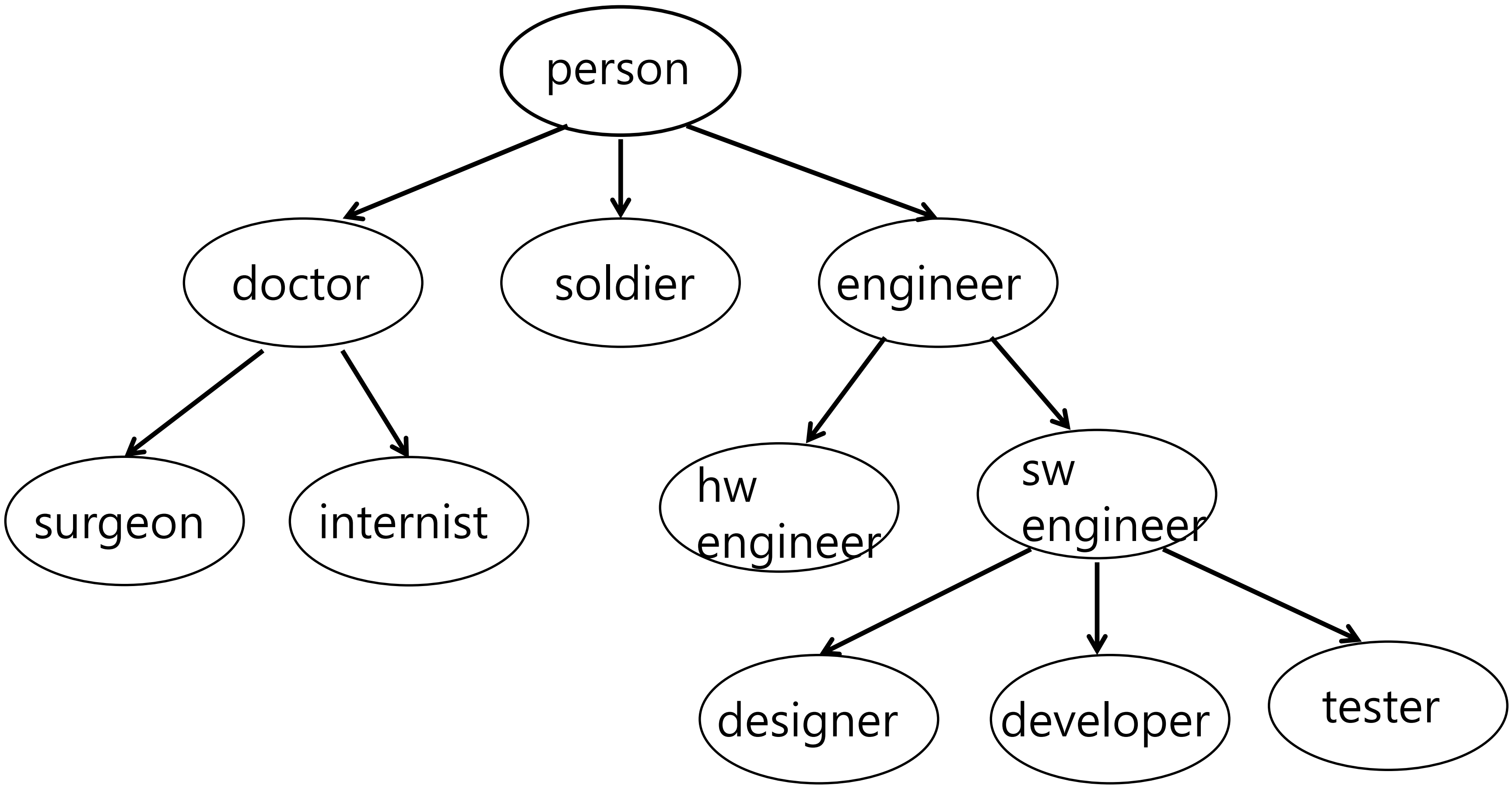
100
75
150
50
90
120
490

Level Order Traversal: Example (4/4)



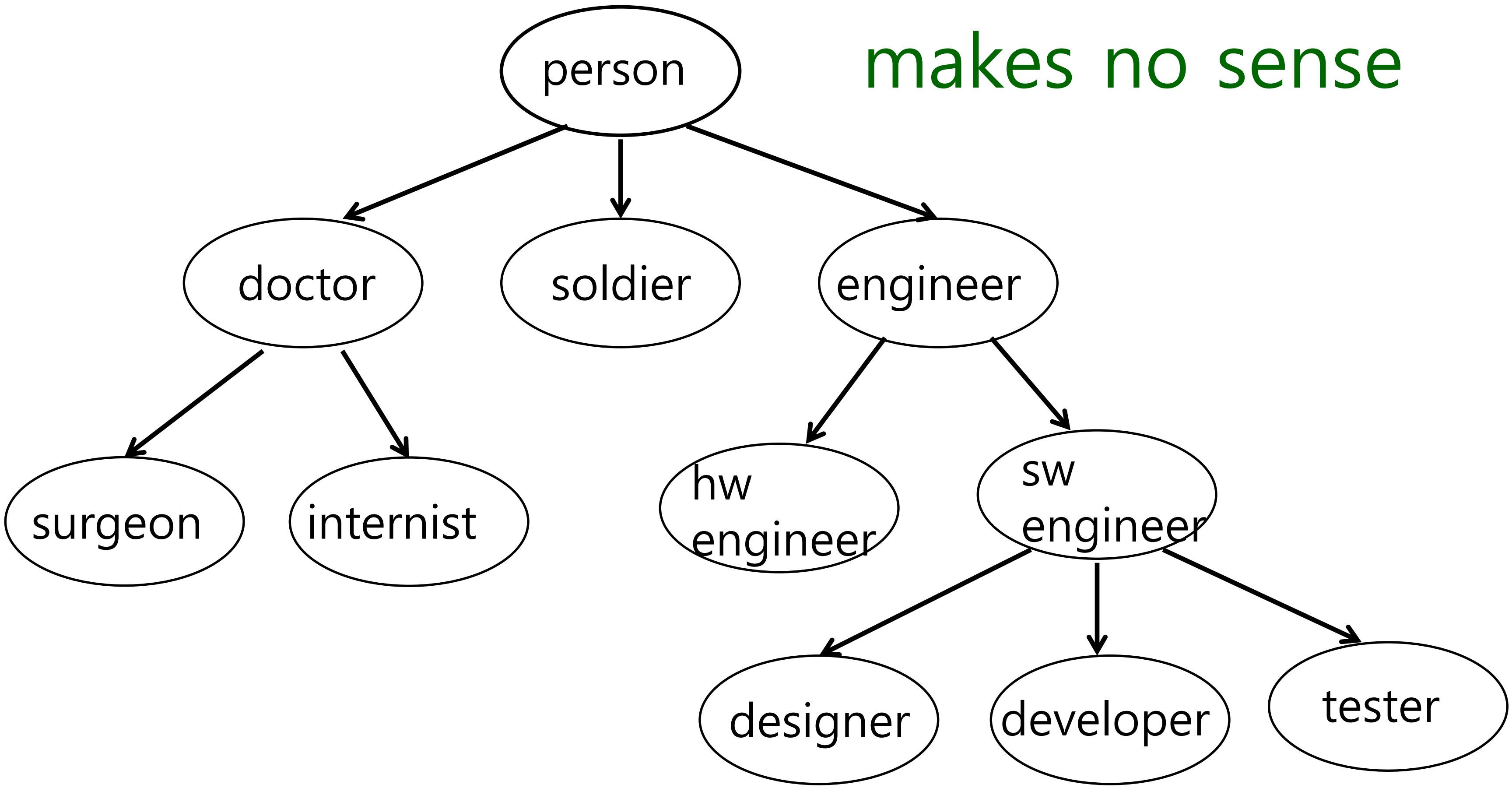
100
75
150
50
90
120
490
160

Inorder Traversal: Exercise

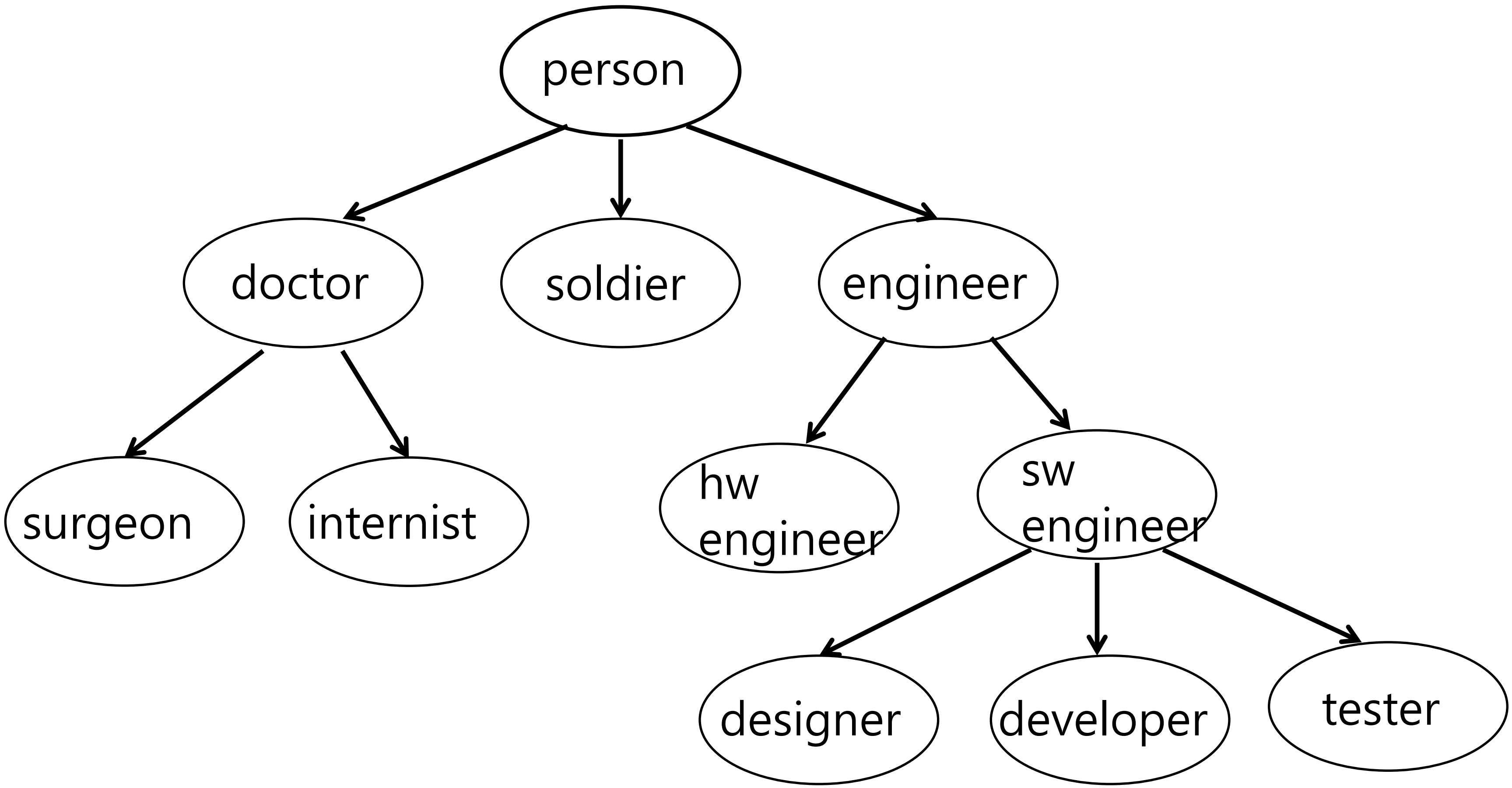


Inorder Traversal: Solution

makes no sense

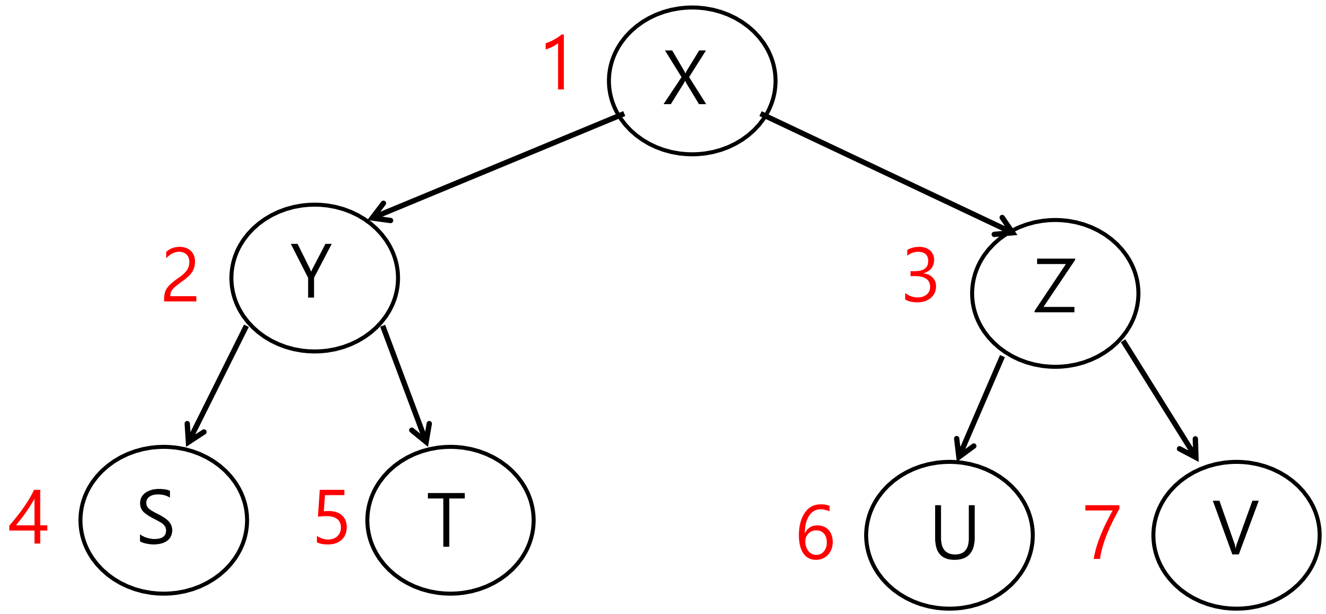


Exercise: What Are the Results of #1 level order, #2 Postorder, #3 Preorder Traversal?

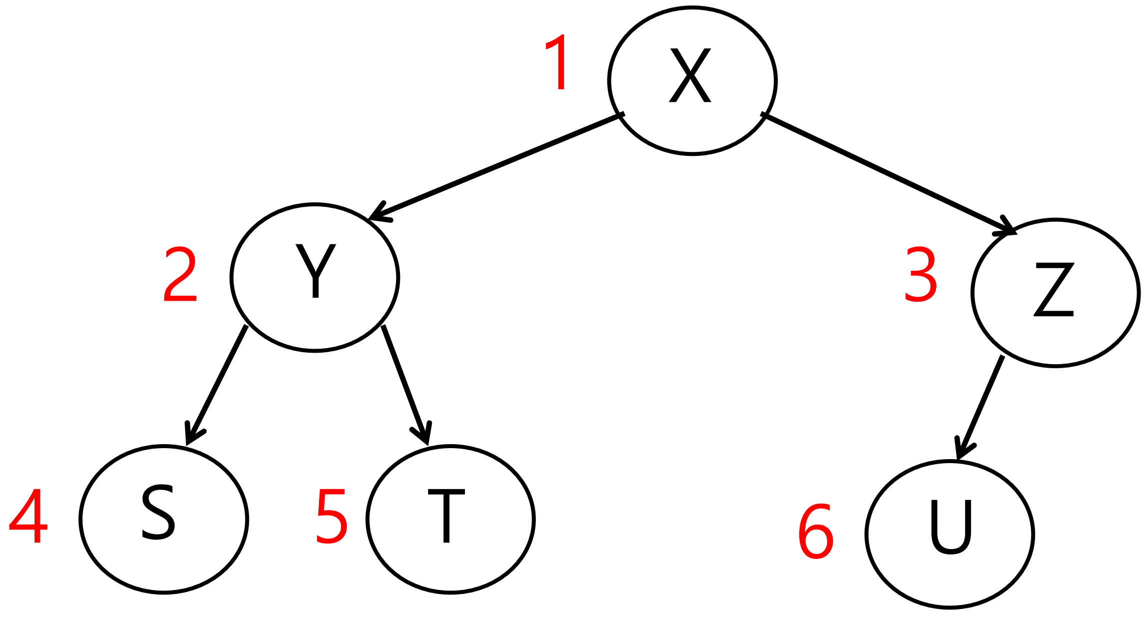


Binary Trees

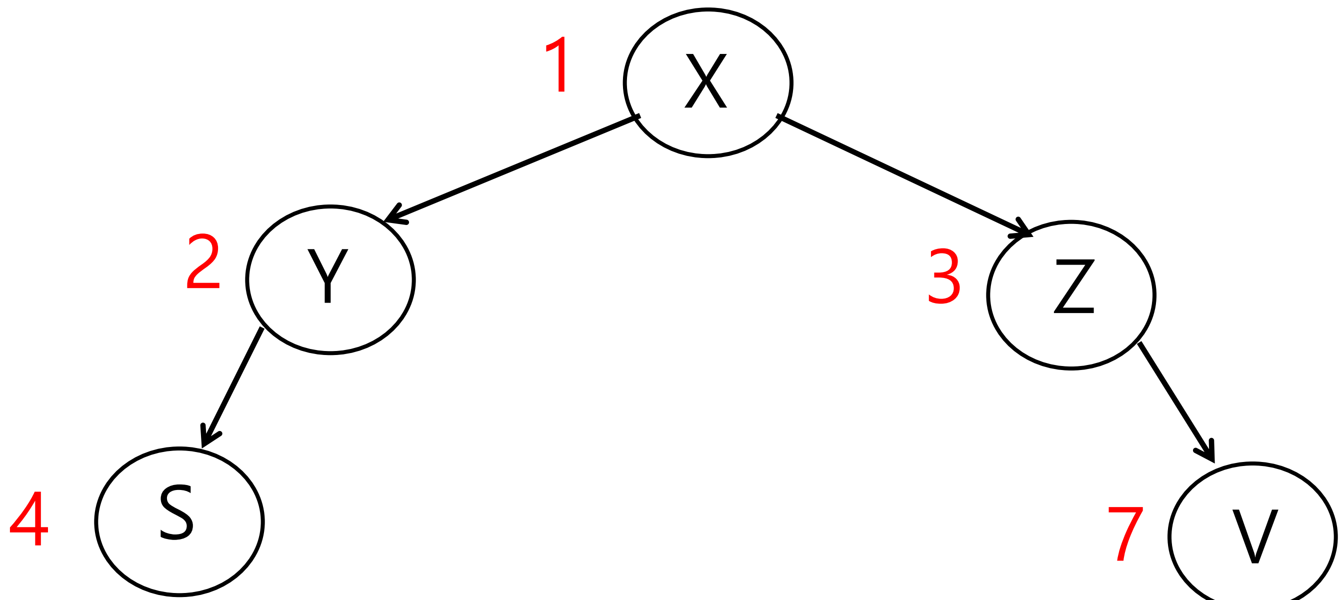
A Full Binary Tree



A Complete Binary Tree

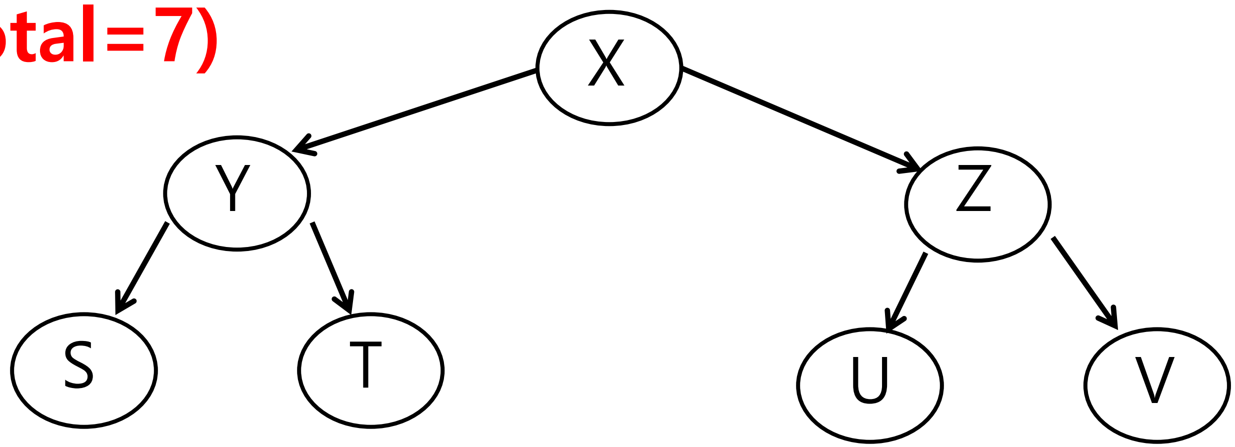


Not a Complete Binary Tree (* middle teeth missing ^ ^ *)

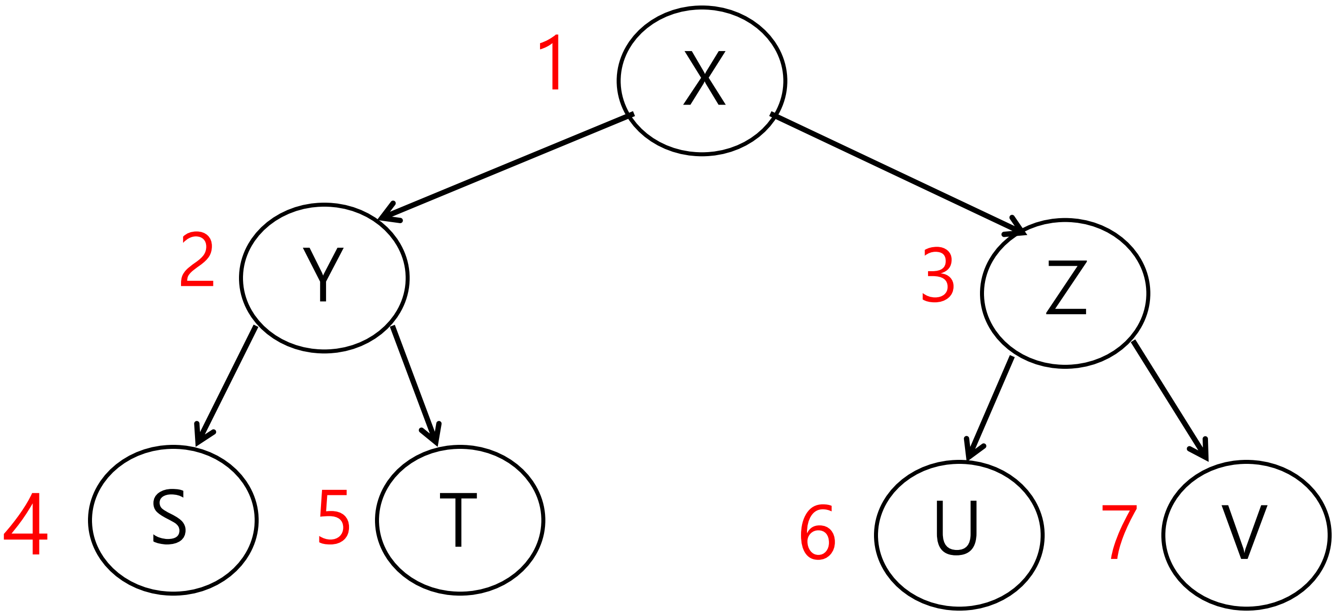


Binary Tree: Properties

- 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)**



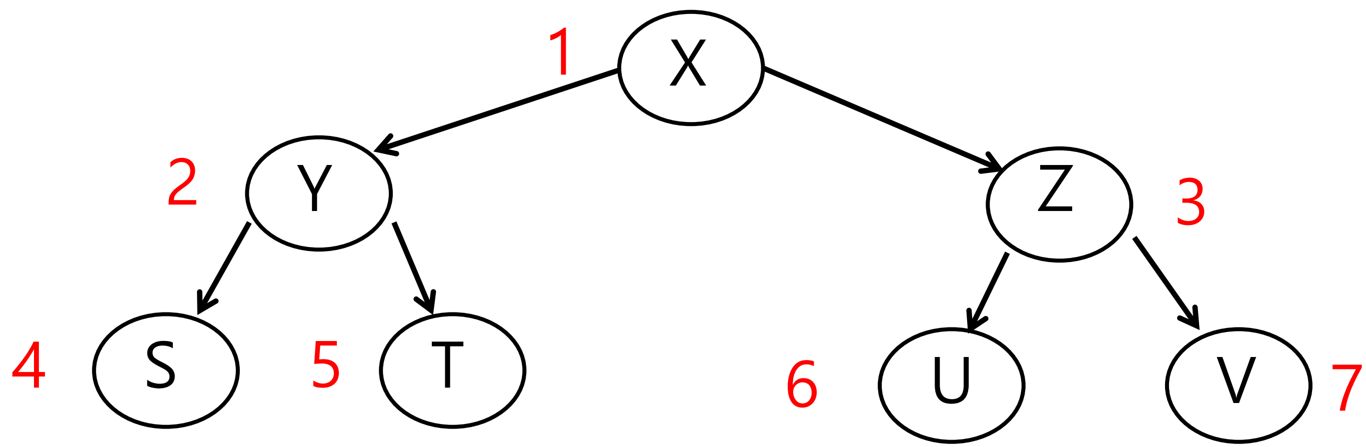
Implementing a Binary Tree Using an Array



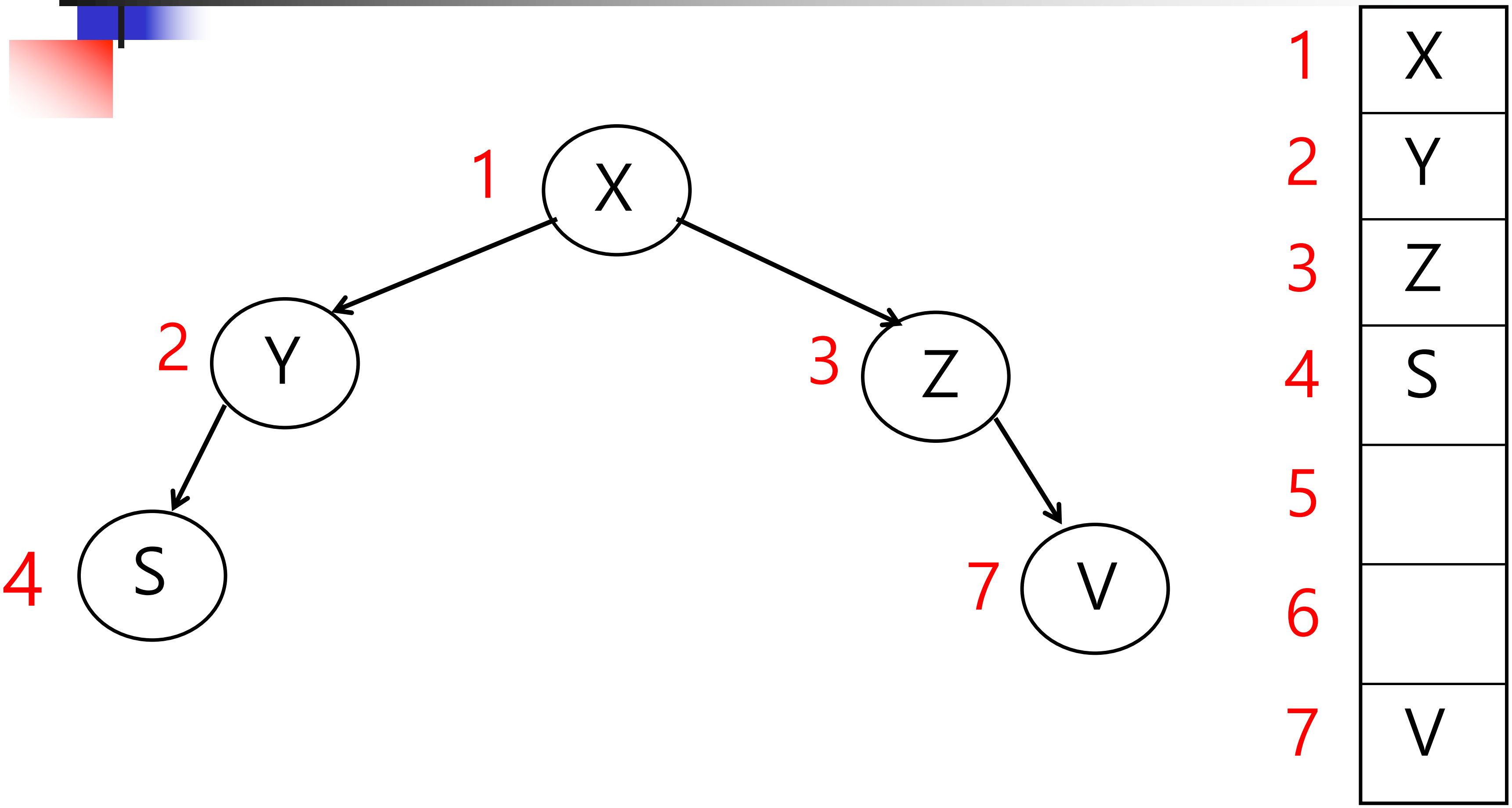
1	X
2	Y
3	Z
4	S
5	T
6	U
7	V

Computing the Locations of Nodes

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



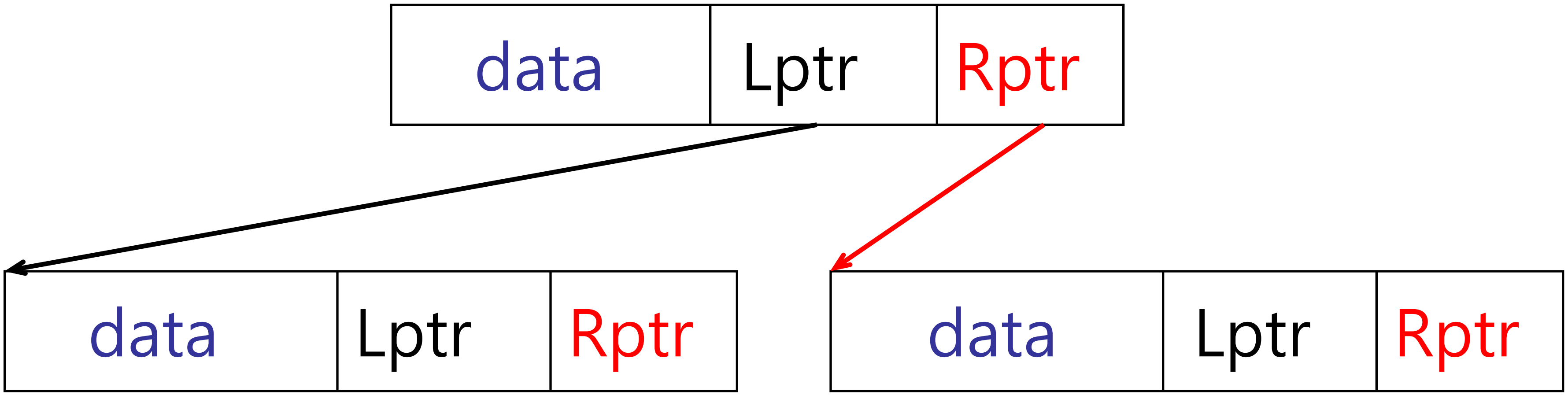
Implementing a Binary Tree Using an Array



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



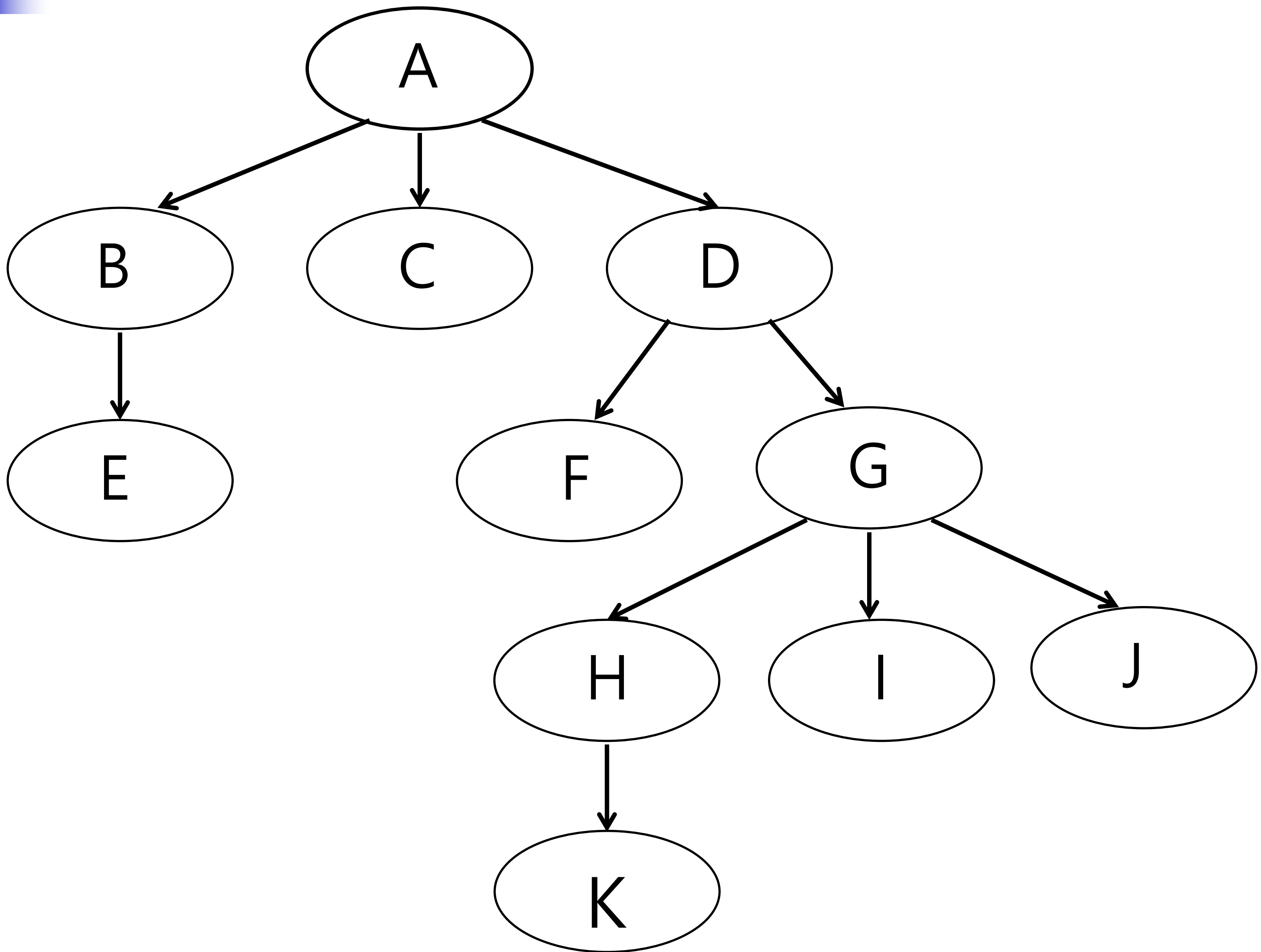
Coding in C

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

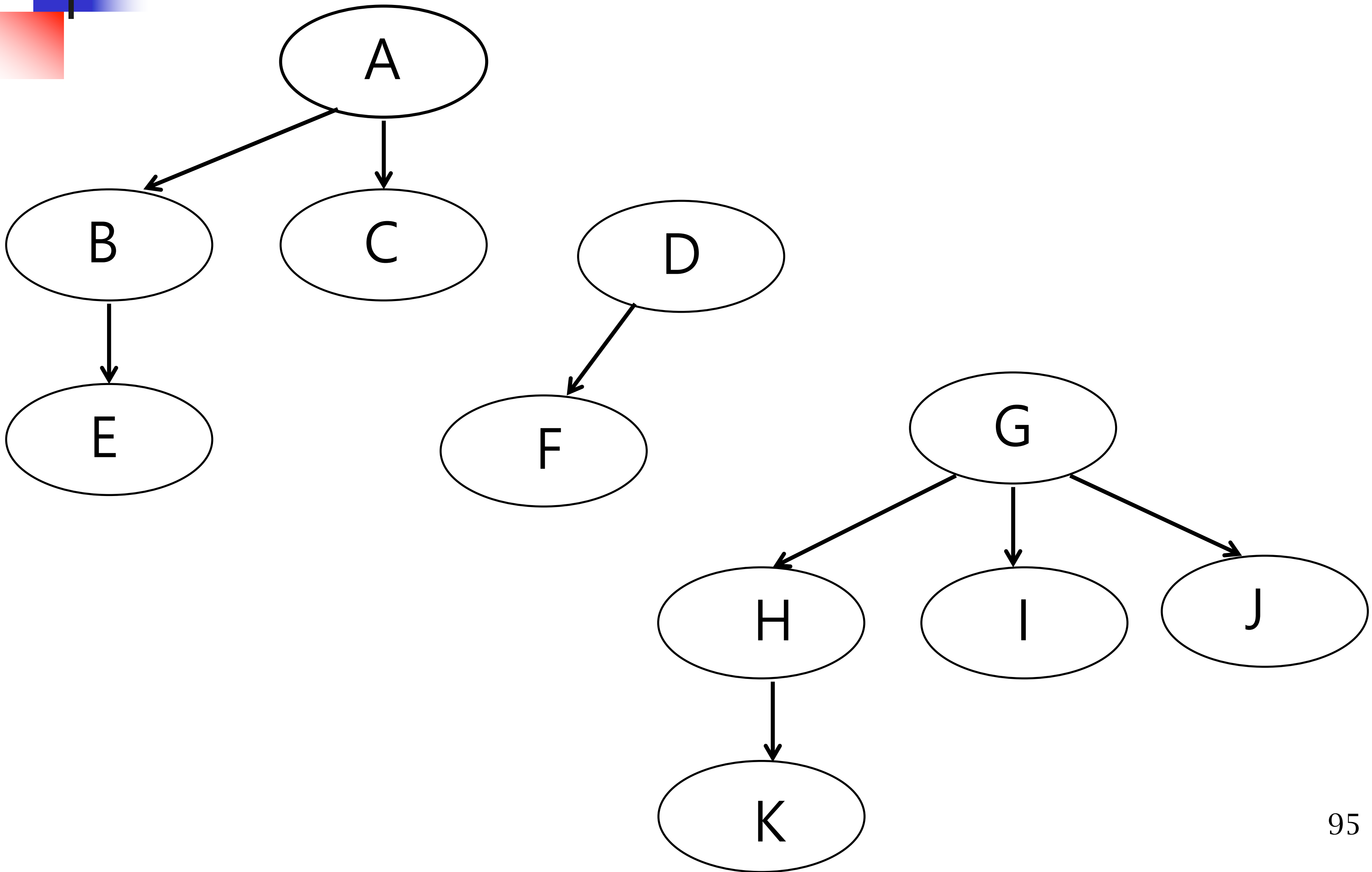
```
typedef  struct  node *tree_ptr;
typedef  struct  node  {
                (data_type    data);
                tree_ptr  left_child, right_child;
                };
```

Representations of Trees

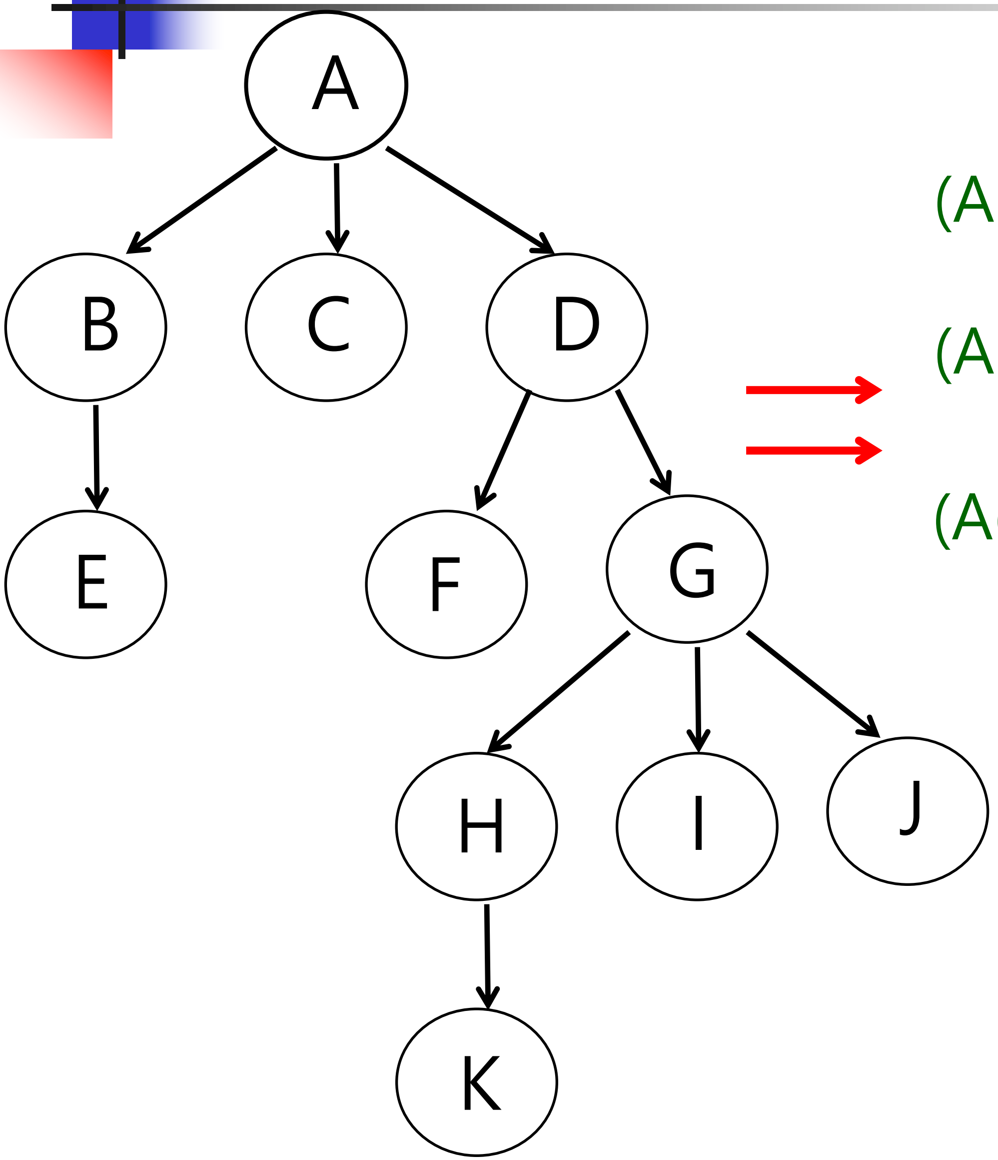
General Tree



Forest (of 3 Trees)



List Representation of a Tree



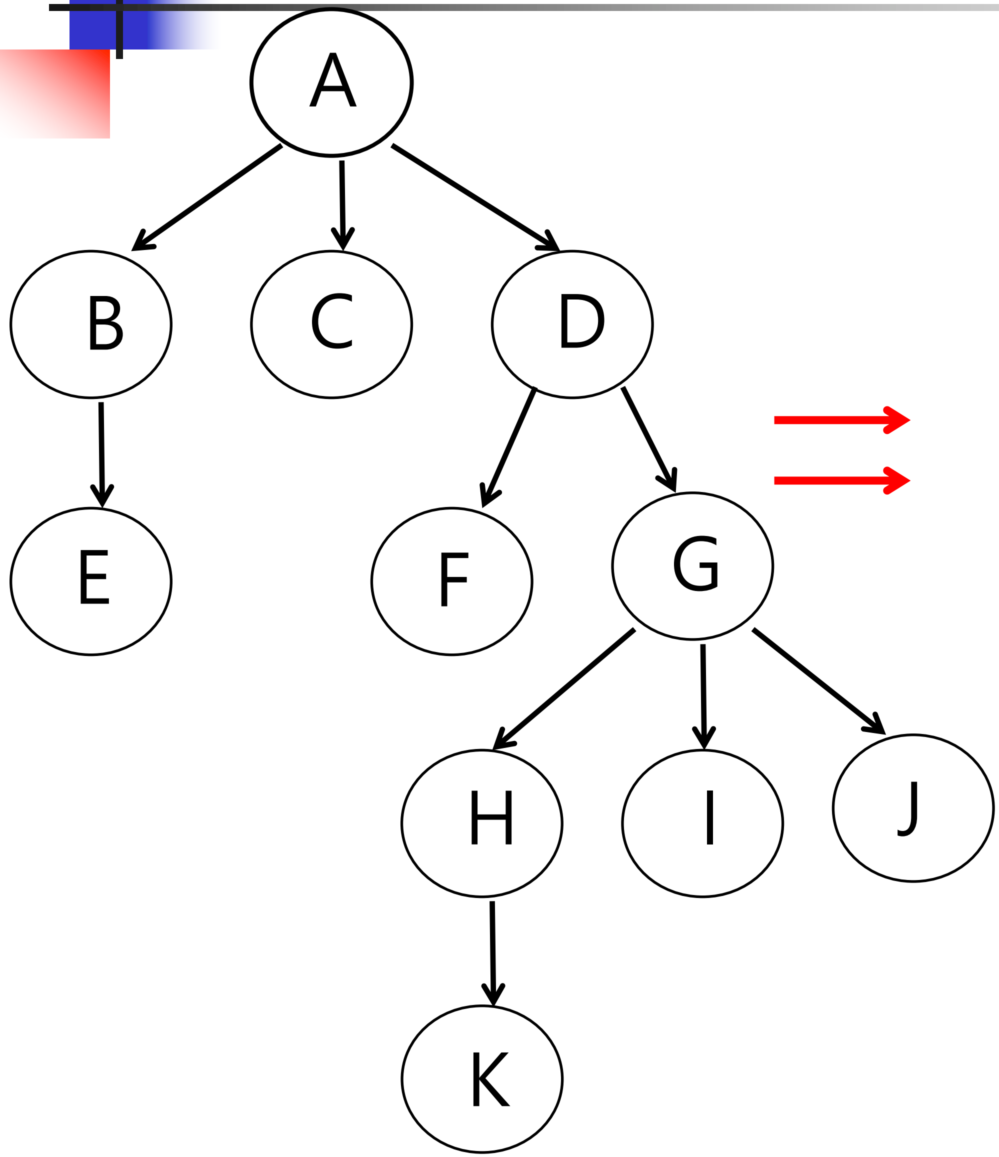
(A(B,C,D))

(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	

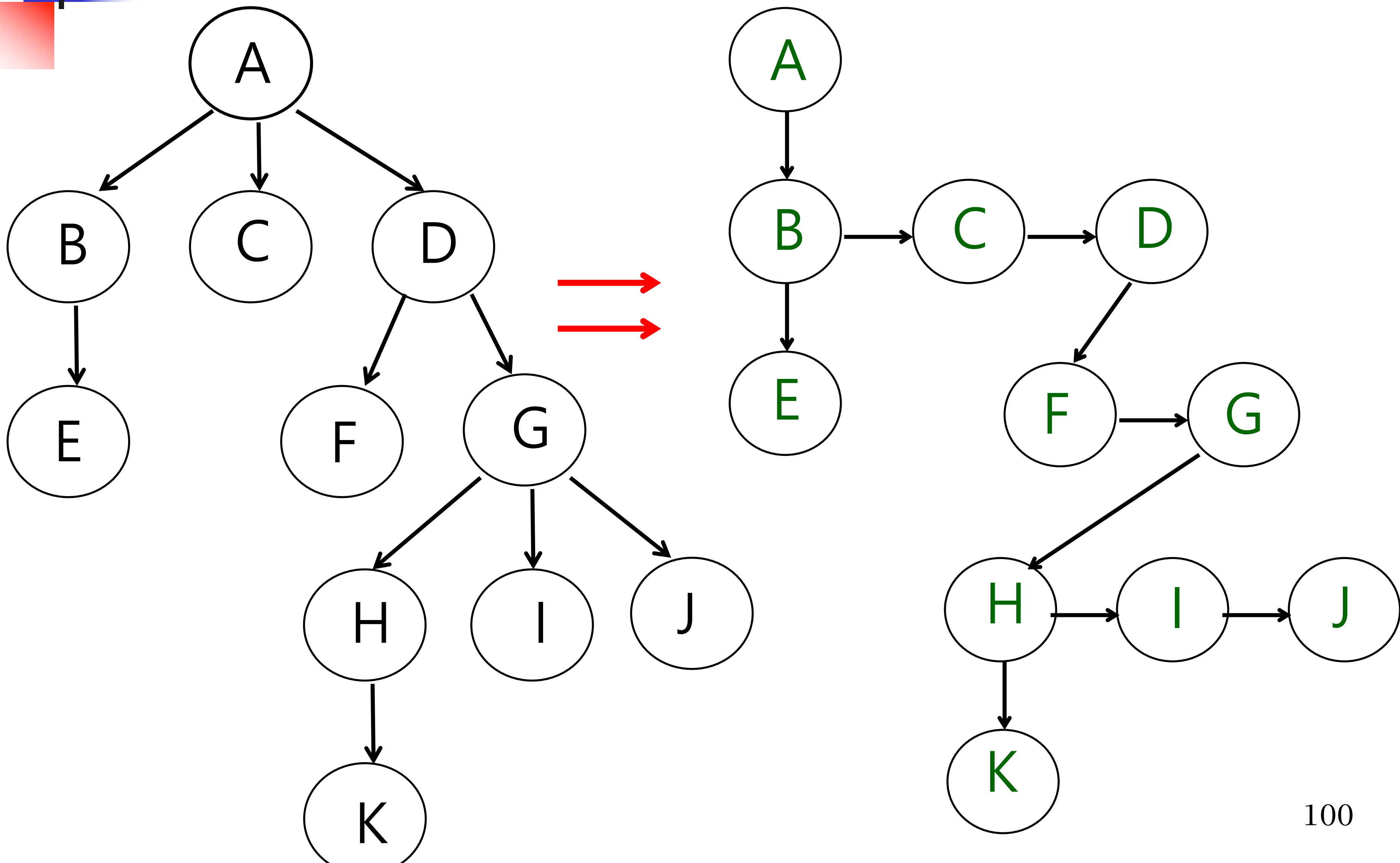
Converting a General Tree into a Binary Tree

- 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.

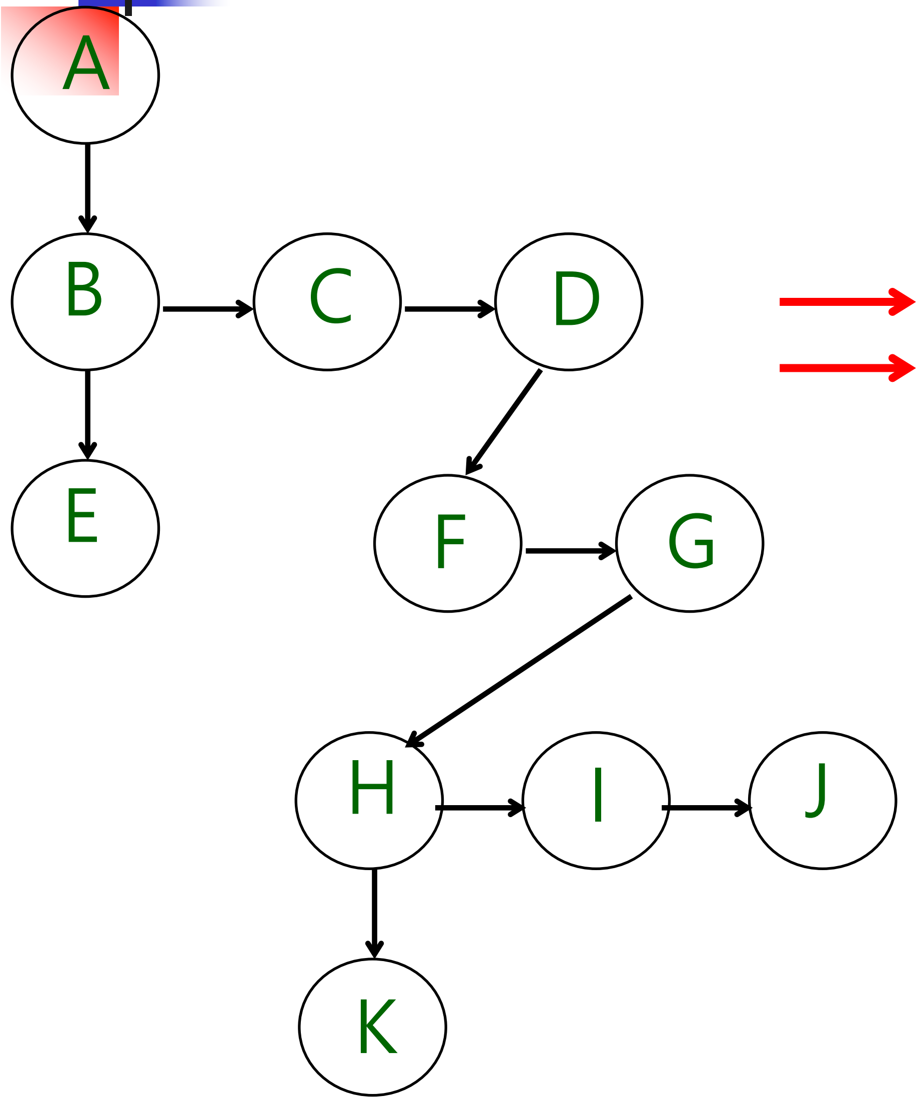
Illustration of Step 1

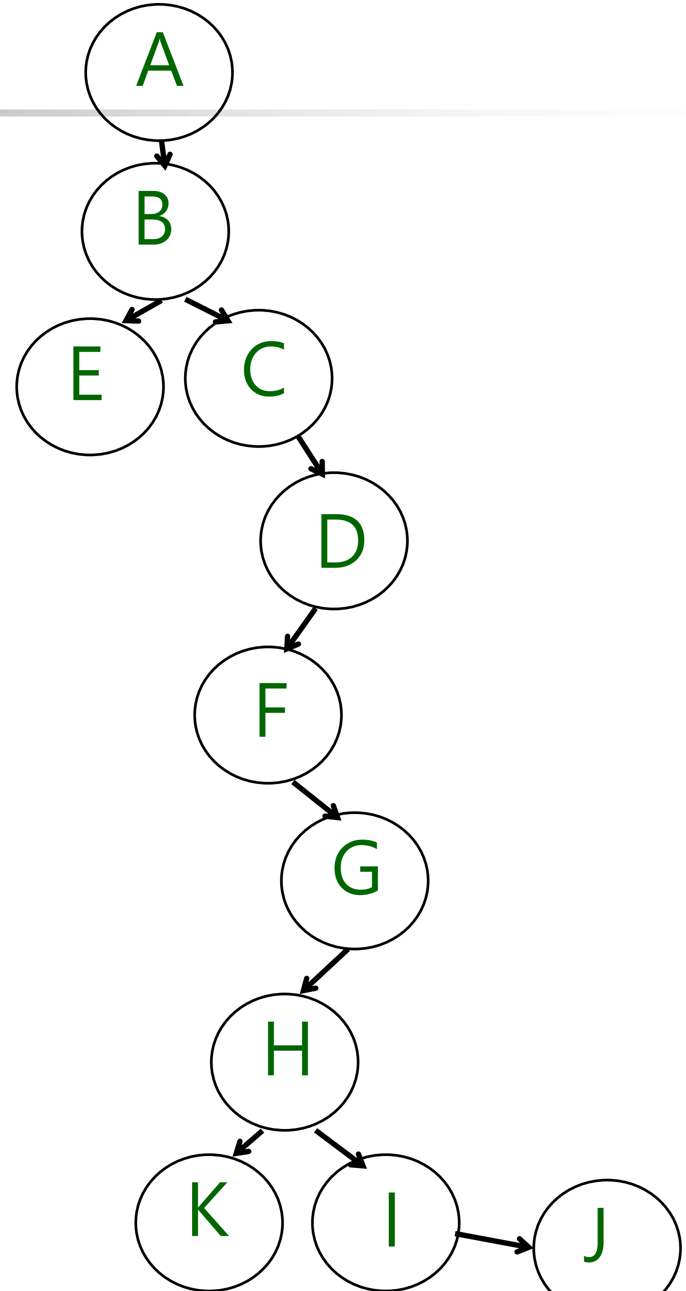


Each Node Has…

data	left child ptr	right sibling ptr

Illustration of Step 2





Each Node Now Has...

data	left child ptr	right child ptr

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)

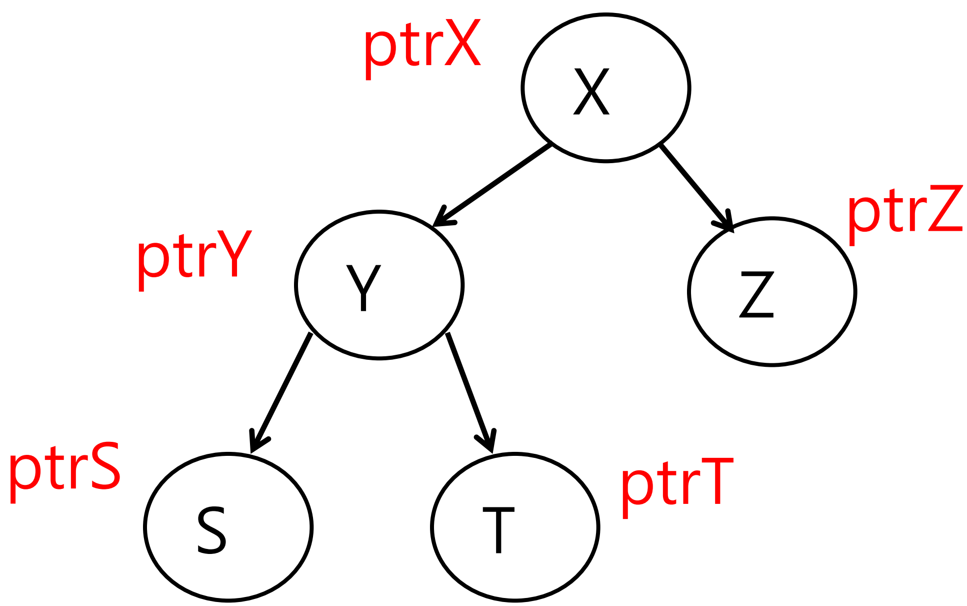
- In the Transformation
  - All nodes of the original general tree are included.
  - Node relative levels are preserved.
  - Parent-descendant relationship is preserved.

Implementing Inorder Traversal in C

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

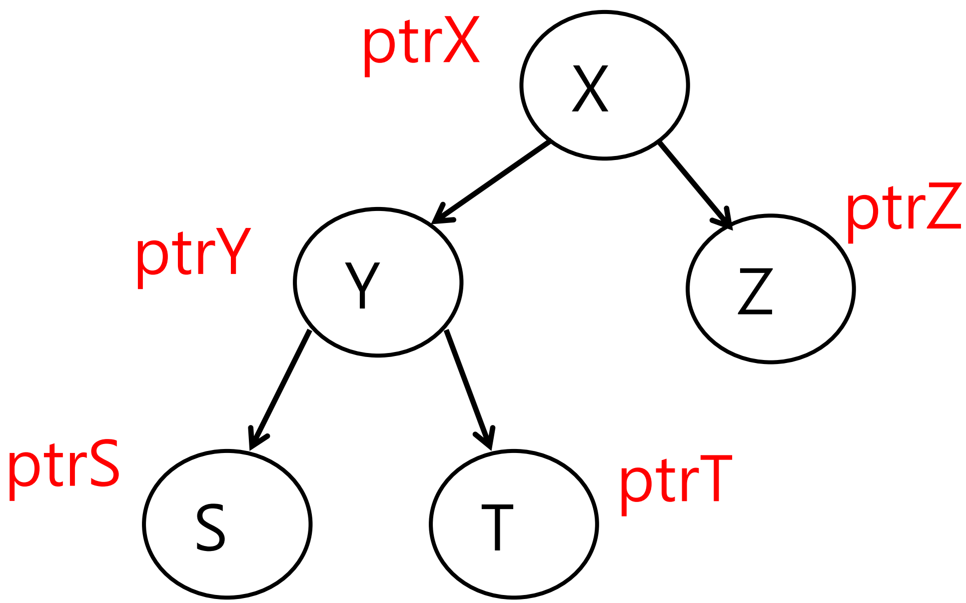
How Recursion Works: (Inorder traversal of a sample tree)

```
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); }
}
```

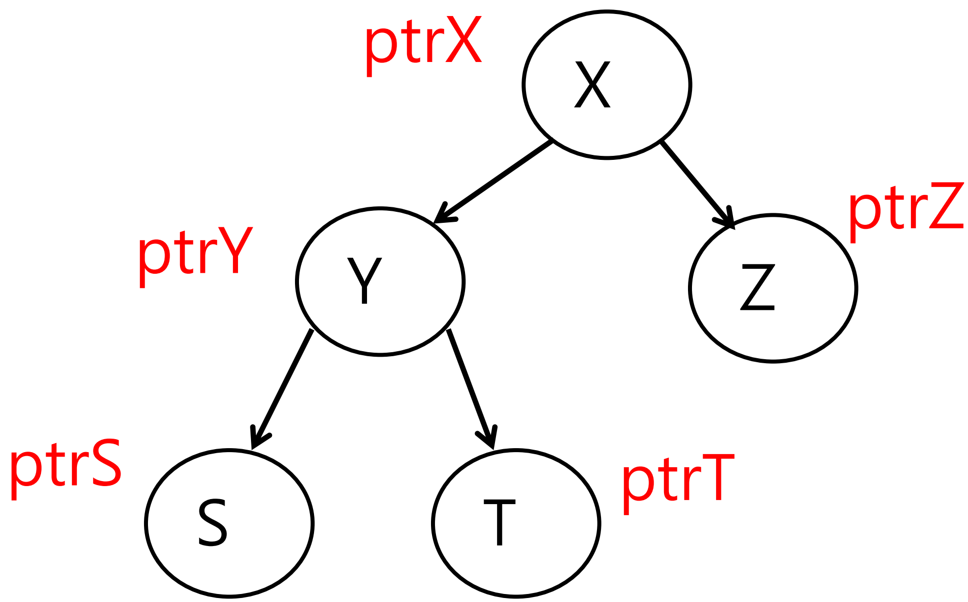


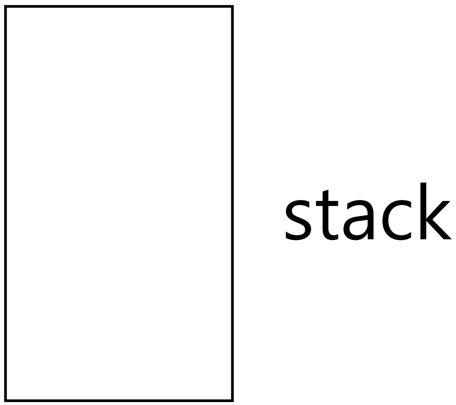

```
inorder (ptrX)
  inorder (ptrY)
    inorder (ptrS)
      inorder (null)
      visit S
      inorder (null)
    visit Y
    inorder (ptrT)
      inorder (null)
      visit T
      inorder (null)
  visit X
  inorder (ptrZ)
    inorder (null)
    visit Z
    inorder (null)
```

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)



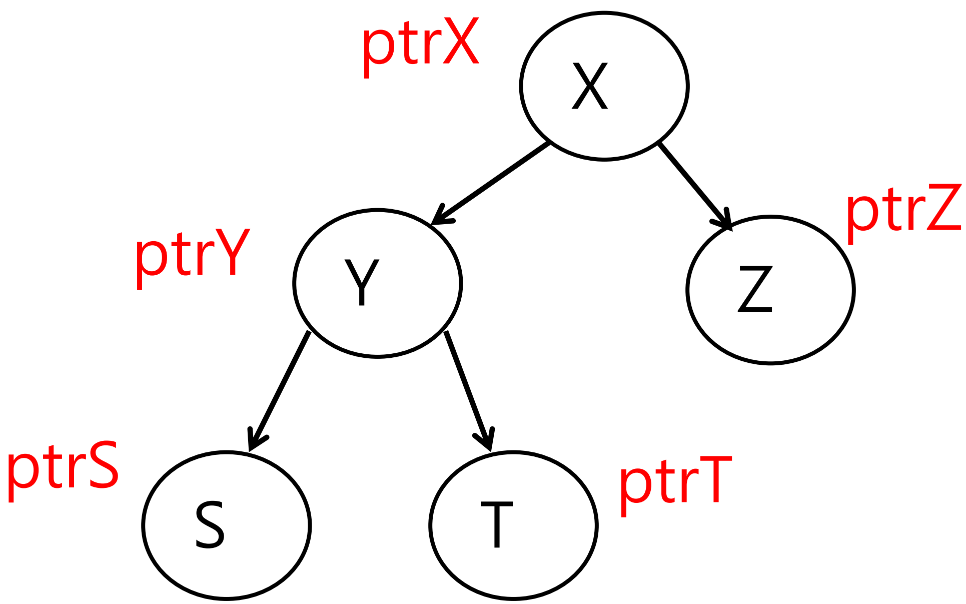


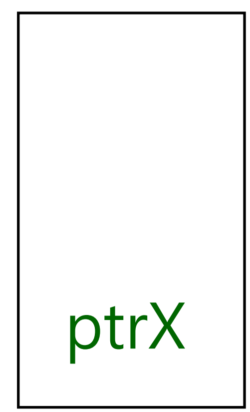
(2/7)

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

inorder (ptrX)

inorder (ptrY)





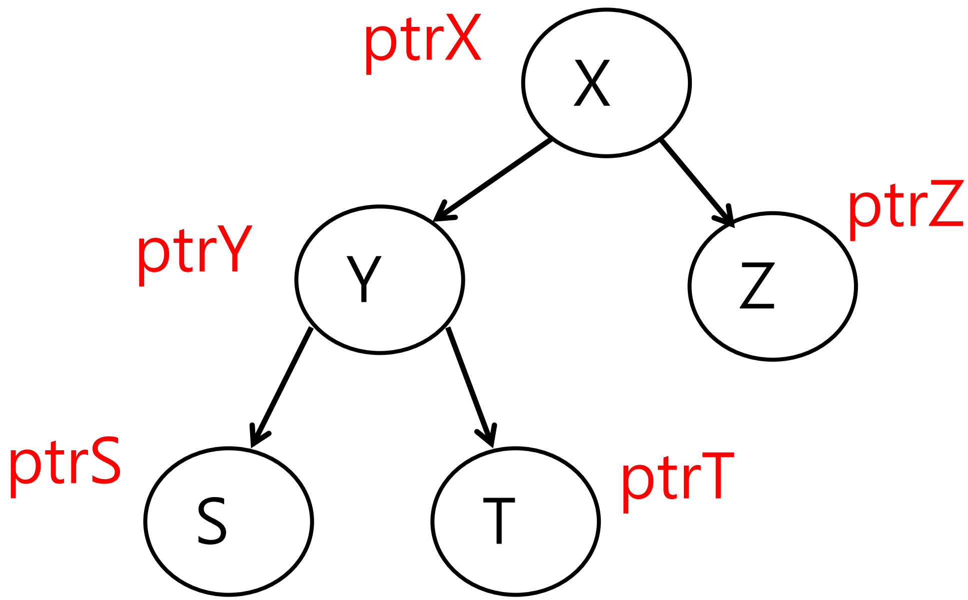
(3/7)

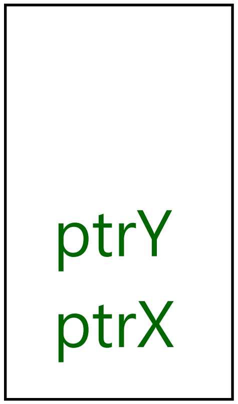
```
void inorder (tree_ptr  ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```

inorder (ptrX)

inorder (ptrY)

inorder (ptrS)





(4/7)

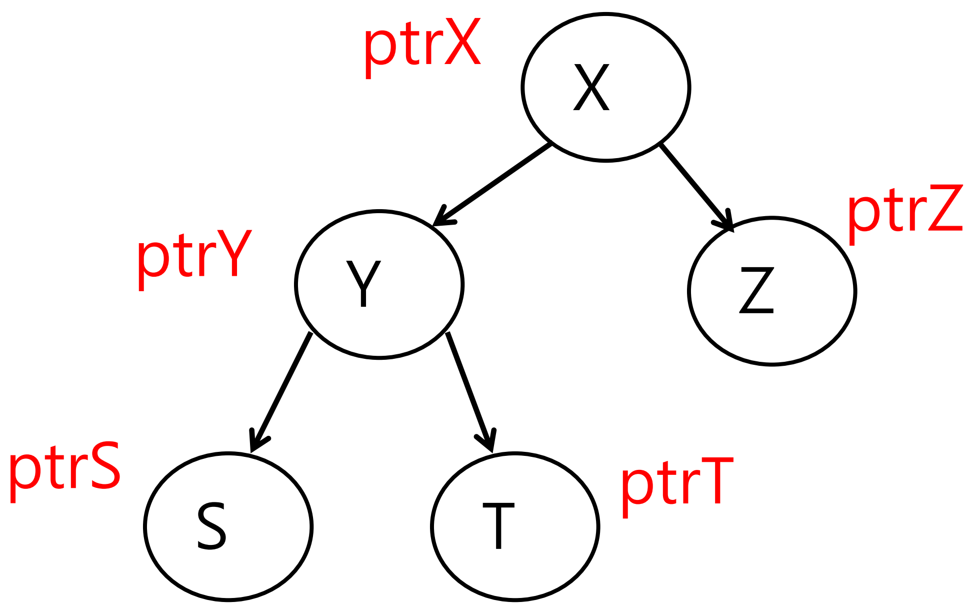
```
void inorder (tree_ptr  ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```

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)

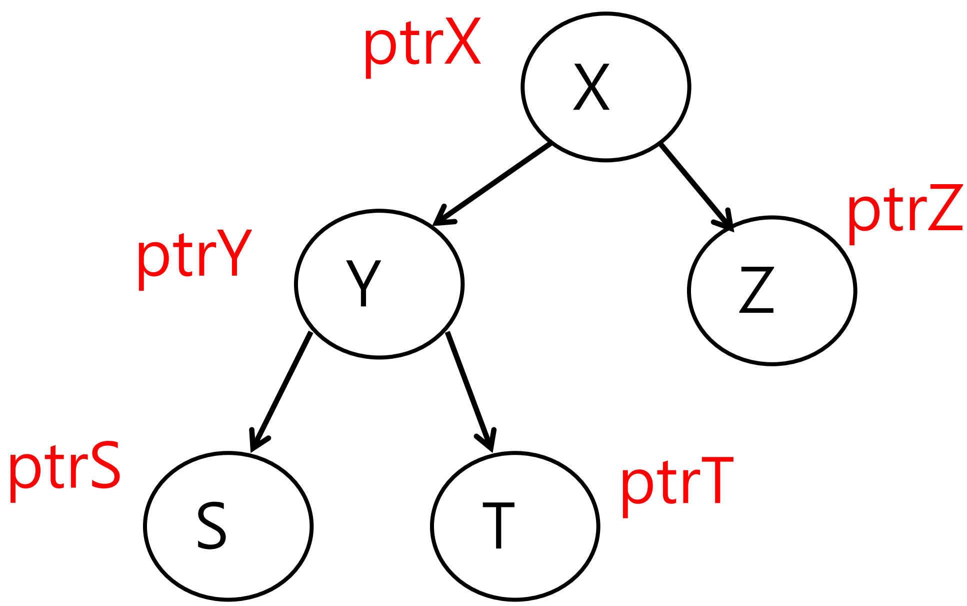
inorder (ptrY)

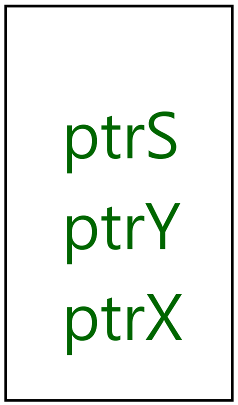
inorder (ptrS)

inorder (Lnull)

visit S

inorder (Rnull)



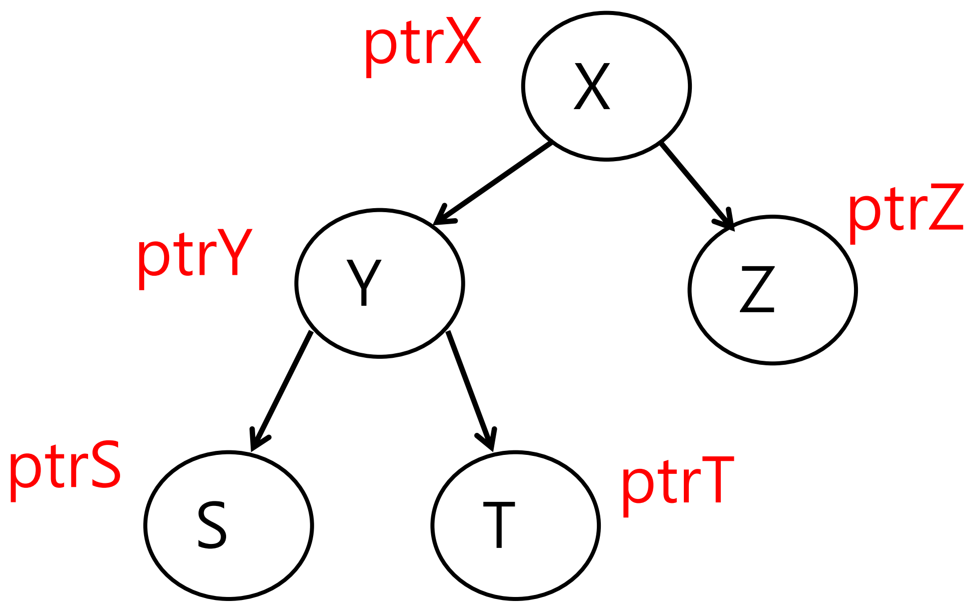


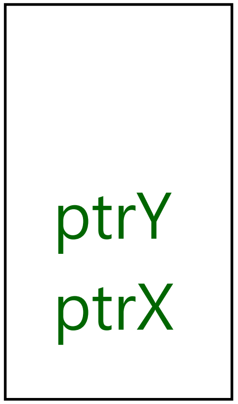
(6/7)

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

inorder (ptrX)

inorder (ptrY)





(7/7)

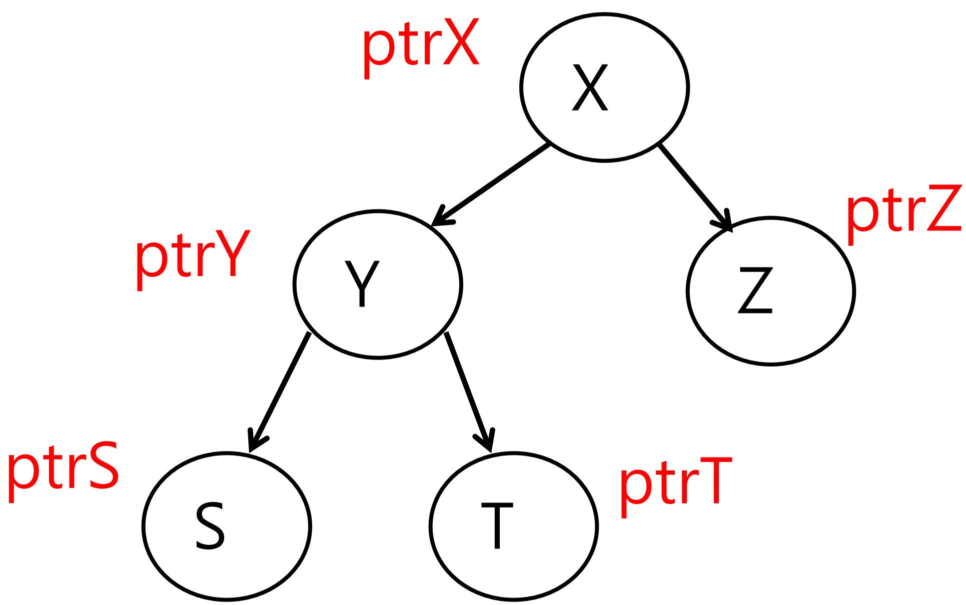
```
void inorder (tree_ptr  ptr)
{ if  (ptr)  {
        inorder (ptr->left_child);
        (visit node);
        inorder (ptr->right_child); }
}
```

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

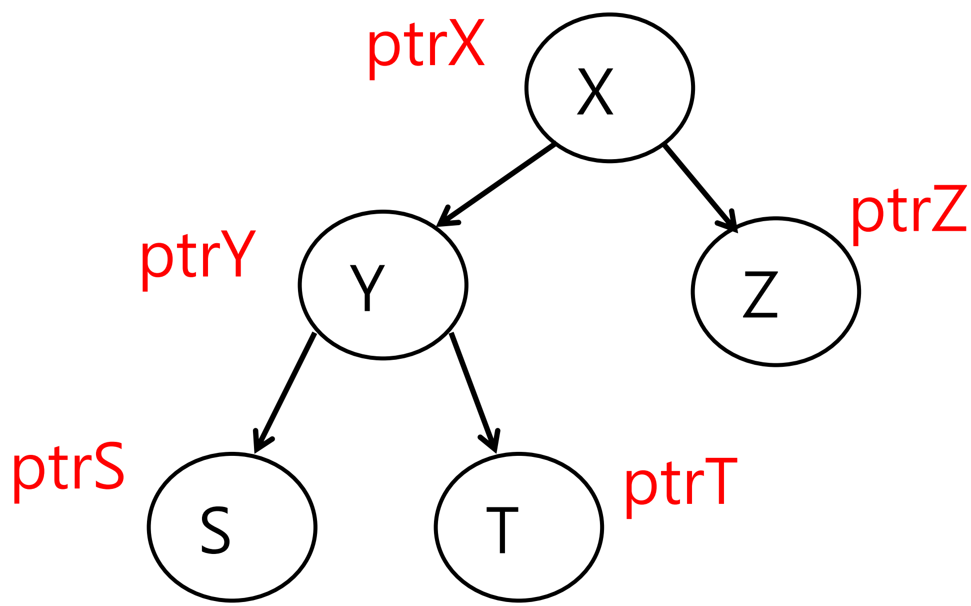
```
void postorder (tree_ptr  ptr)
{
    if  (ptr)  {
        postorder (ptr->left_child);
        postorder (ptr->right_child);
        (visit node);
     }
}
```

Implementing Preorder Traversal in C

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

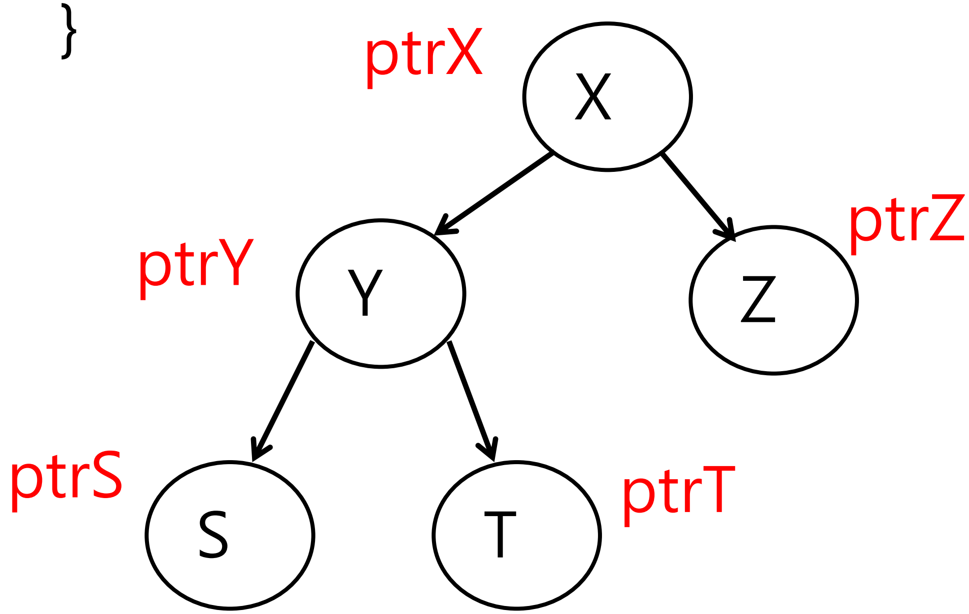
WHW 1-1: Draw the Recursion Execution Sequence and Stack State Sequence (10 points)

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



WHW 1-2. (5 points) Draw the Recursion Execution Sequence

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



Programming Homework (PHW) 2

- Two problems
- Specifications in a separate WORD file.

End of Lecture

Review of
Programming Homework 1

Review: malloc and free functions

```
int *nums, i;

nums = (int *) malloc (10*sizeof(int));
if (nums == (int *) NULL) {
    printf (“malloc failed”);
    exit(1);
    }
for (i=0; i<10; i++) {
    printf (“₩n type an integer”)
    scanf (“%d”, &nums[i]);
    printf (“%d”, nums[i]);
    }
free(nums);
```

Review: malloc and free functions

```
int *nums, i;

nums = (int *) malloc (10*sizeof(int));
if (nums == (int *) NULL) {
    printf ("malloc failed");
    exit(1);
    }
for (i=0; i<10; i++) {
    printf ("₩n type an integer")
    scanf ("%d", &nums[i]);
    printf ("%d", nums[i]);
    }
free(nums);
```

Review: Programming Homework 1

- Implementing a stack or queue using a singly linked list
- (WRONG) It makes no sense to test for stack_full, queue_full condition
  - A linked list has infinite memory. All you need to do is call malloc and request additional memory.
- If this is the case, why is there malloc in the first place?

Why Is Memory Not Infinite?

- Computer main memory is taken up by the operating system and other programs and the buffer space they need.
- Your program is given limited space (by the operating system).
- On top of that, there are memory leaks.

Memory Leak

- Imperfect programming causes memory leaks – many large programs have them.
- A **memory leak** occurs when a computer program does not free memory that is no needed.
  - no 'free' after 'malloc'
- A **memory leak** also occurs when an object stored in **memory** cannot be accessed
  - (e.g.) node A has a pointer to node B; the pointer is deleted.

stack_full, queue_full testing

- defensive coding by checking for malloc failure.

```
int *nums, i;

nums = (int *) malloc (10*sizeof(int));
if (nums == (int *) NULL) {
    (malloc failed);
  }
```

Damages Done by Careless Memory Management

- Not checking for malloc failure
- Not freeing memory no longer needed
- Lead to system crashes for large important programs
- A lot of time, people and money spent to debug and re-release the programs