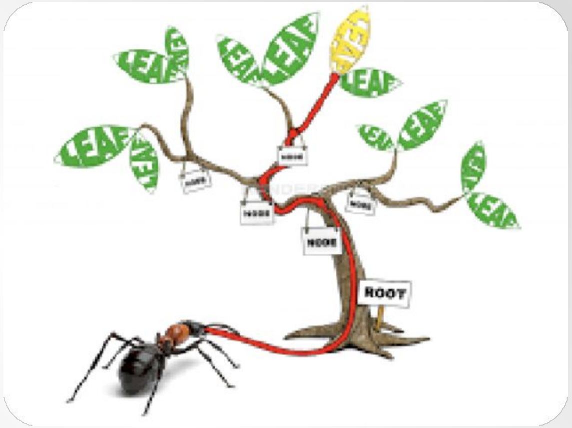
Data structure and Algorithms

Presented by : Asmaa Ghonaim

Trees

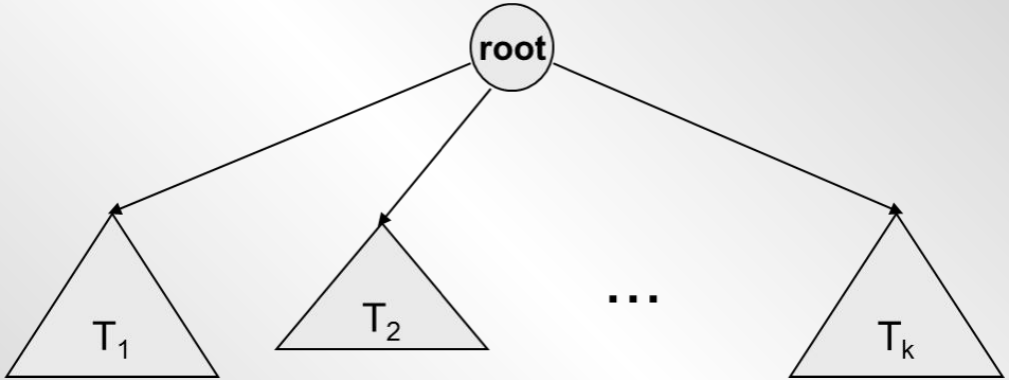


Trees

A tree is a collection of nodes with the following properties:

- The collection can be empty.
- Otherwise, a tree consists of a distinguished node r , called root, and zero or more non empty sub-trees T1, T2, … , Tk, each of whose roots are connected by a directed edge[path] from r.

- The root of each sub-tree is said to be child of r, and r is the parent of each sub-tree root.



Trees

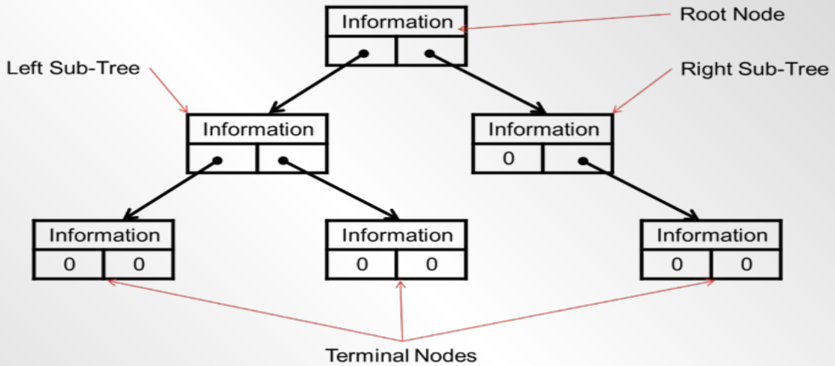
Terminology of Trees:

- Node (also called a Leaf): is any data item in the tree.
- Root Node(also called Parent Node) : is the first item in the tree.
- Subtree: is any piece (i.e., branch) of thy tree.
- Terminal Node: is a node that has no subtrees attached to it.
- Tree Height: is equal to the number of layers deep that its root grows.
- If a tree is a collection of N nodes, then it has N-1 edges.

N nodes = **6**

N – 1 edges= **5**

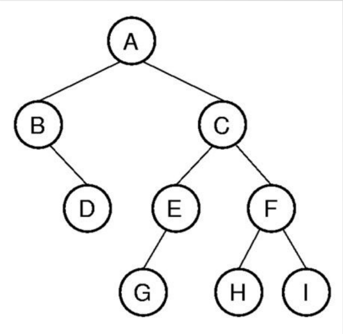
Tree Height = **3**



Trees

Terminology of Trees:

- A path from node n_1 to n_k is defined as a sequence of nodes n_1, n_2, ..., n_k such that ni is parent of n_{i+1} $(1 \le i < k)$
- The length of a path is the number of direct edges on that path.
- There is a path of length zero from every node to itself.
- There is exactly one path from the root to each node.
- The depth of node ni is the length of the path from root to node n_i
- The height of node ni is the length of longest path from node ni to a leaf.



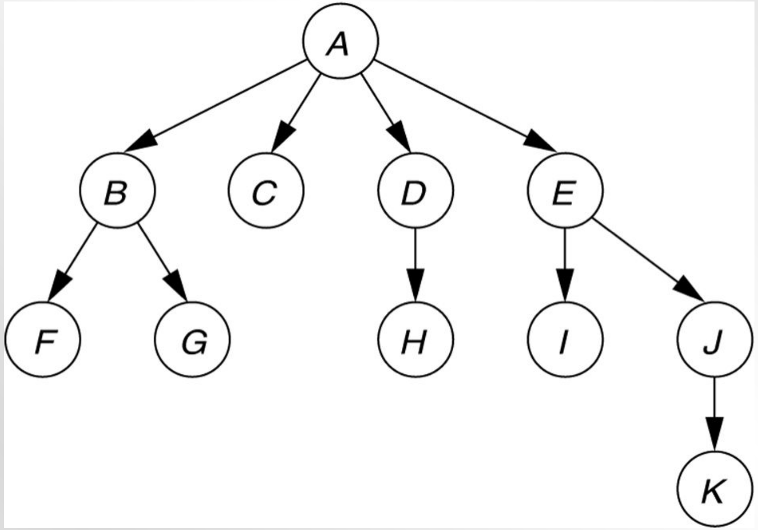
Trees

➔ A tree, with height and depth information for each Node

N nodes = **11**

N – 1 edges= **10**

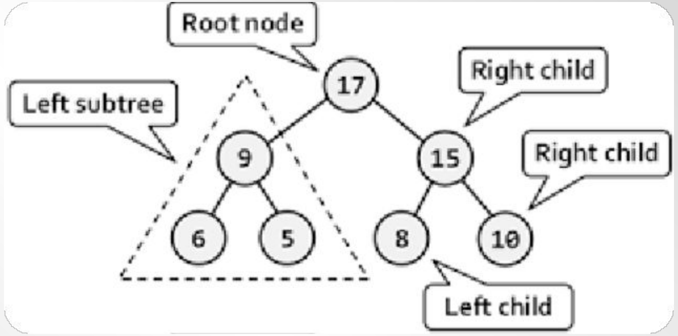
Tree Height = **4**



Node	Height	Depth
A	3	0
B	1	1
C	0	1
D	1	1
E	2	1
F	0	2
G	0	2
H	0	2
I	0	2
J	1	2
K	0	3

Binary Trees

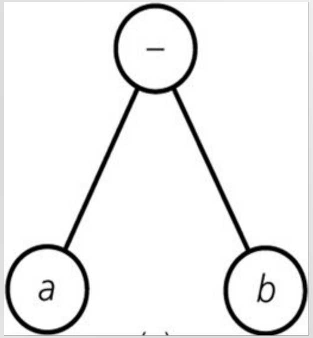
- Binary Trees are special type of Trees because, when they are sorted, they lend themselves to rapid searches, insertions, and deletions.
- Binary Tree is a non-linear and hierarchical data structure where each node has at most two children referred to as the left child and the right child.
- The Binary Tree is a special form of Linked List:
  - Items can be inserted, deleted, and accessed in any order.
  - Also, the retrieval operation is nondestructive.



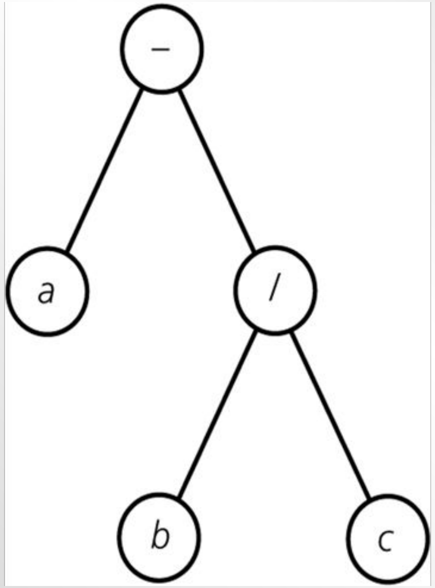
Binary Trees

Binary Tree – Representing Algebraic Expressions:

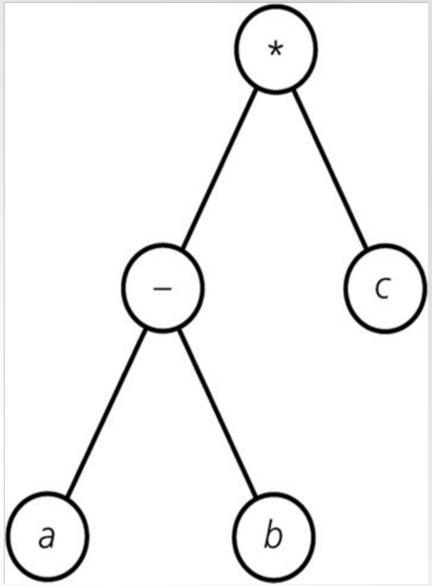
a – b



a – b/c

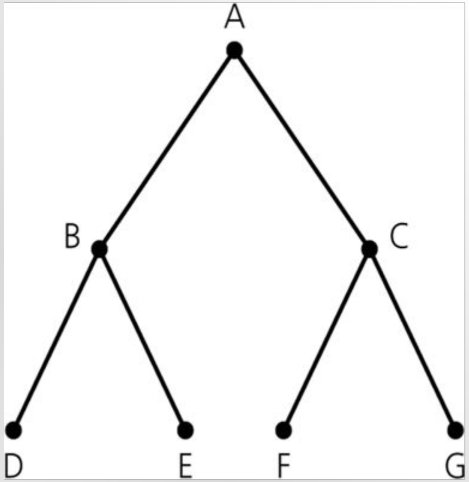


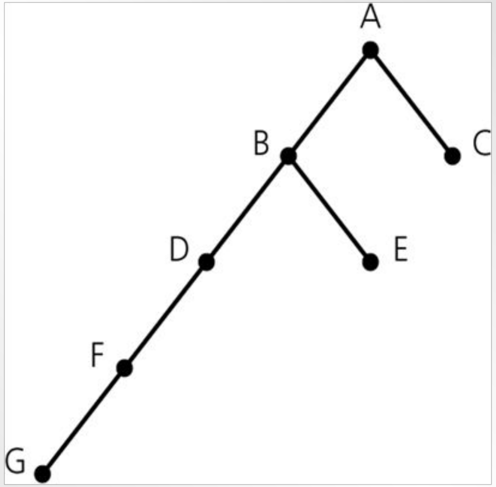
(a – b) * c

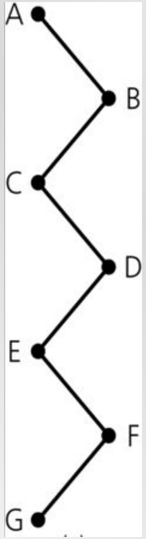


Binary Trees

Height of Binary Tree





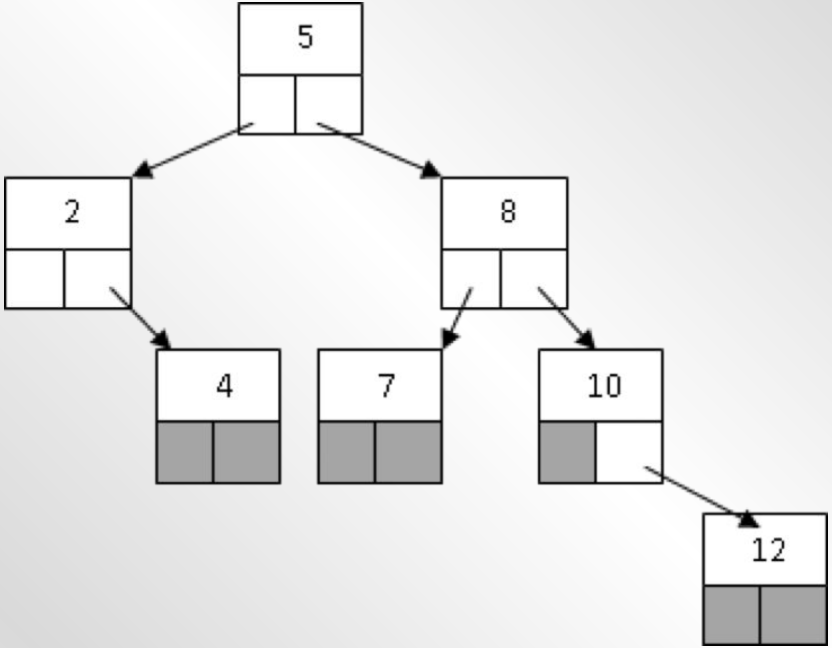


3 5 7

Binary trees with the same nodes but different heights

Binary Trees

Represent these values in binary tree (5, 8, 2, 4, 10, 12, 7)



Binary Trees

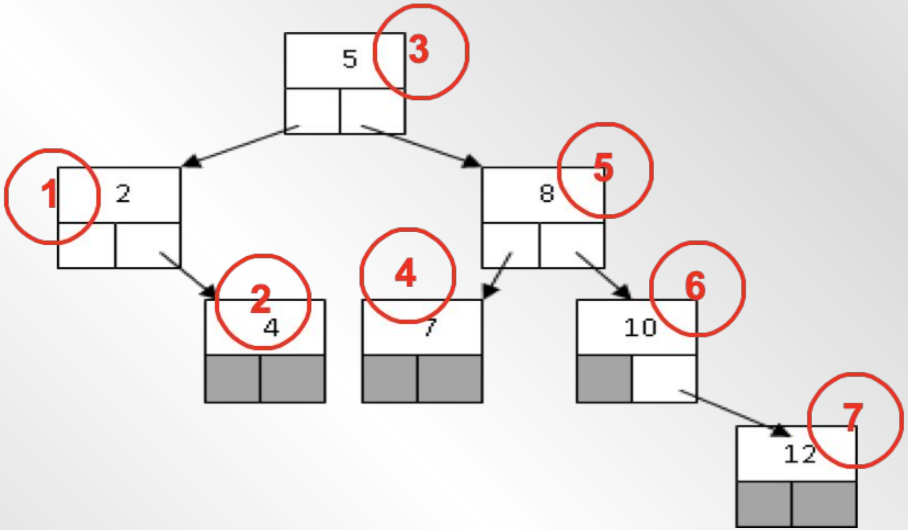
- Traversing Binary Trees:
  - is the process of accessing each node in a tree
- Depth-first search:
  - How a tree is ordered depends on how it is going to be accessed. Generally, there are three ways to traverse a tree:
  - Inorder: you visit the left subtree, the root, and then the right subtree.
  - Preorder: you visit the root, the left subtree and then the right subtree.
  - Postorder: you visit the left subtree, the right subtree, and then the root.
- **Sorted Binary Tree**: inorder traversing is one where the subtree on the left contains nodes that are less than or equal to the root, and those on the right are greater than the root.

Binary Trees

➔ Traversing Binary Trees:

- Inorder: you visit the left subtree, the root, and then the right subtree.

2,4,5,7,8,10,12

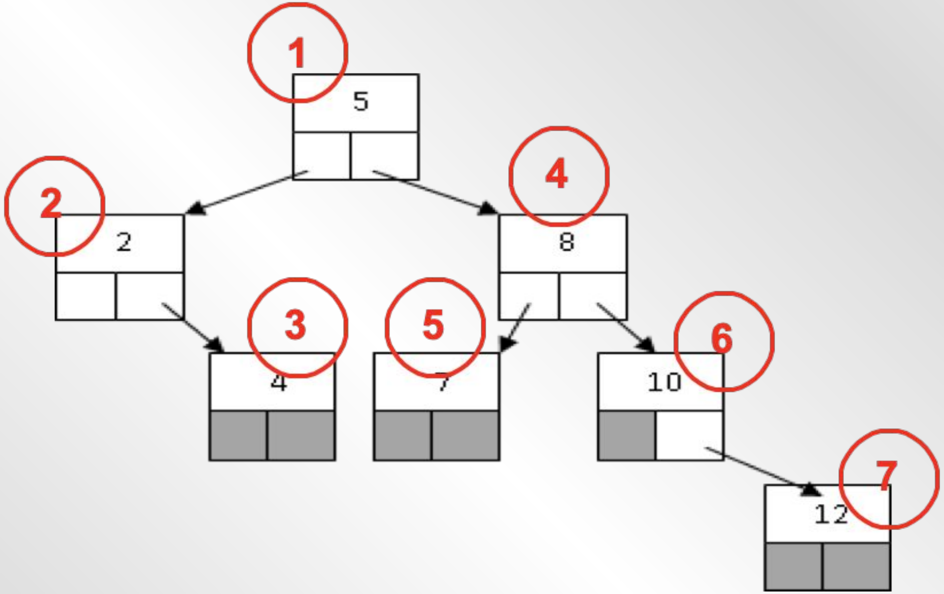


Binary Trees

➔ Traversing Binary Trees:

- Preorder: you visit the root, the left subtree and then the right subtree

5,2,4,8,7,10,12

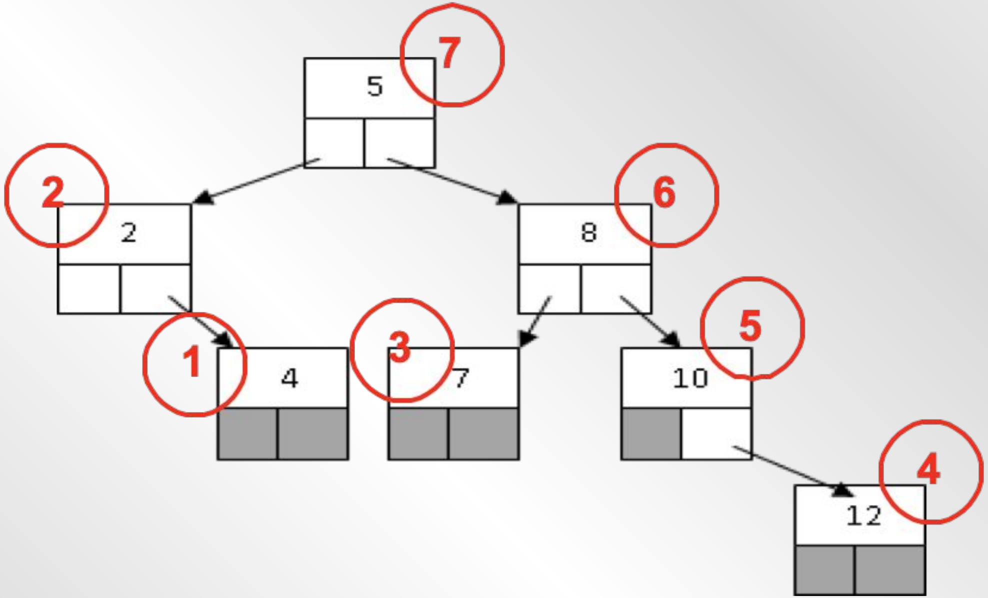


Binary Trees

➔ Traversing Binary Trees:

- Postorder: you visit the left subtree, the right subtree, and then the root.

4,2,7,12,10,8,5



Binary Trees

➔ Operations of Binary Tree

- Creating a Tree (Inserting New Nodes)
    - Case 1: insert root node.
    - Case 2: insert a leaf node.
- Traversing the Tree
    - Method 1: Inorder.
    - Method 2: Preorder.
    - Method 3: Postorder.
- Searching for a particular Node.
- Tree Height
- Deleting Nodes
    - Case 1: delete a leaf node.
    - Case 2: delete a node with one subtree.
    - Case 3: delete a node with two subtree

Binary Trees

Building a Binary tree using Employee as tree node:

```
class Employee {
    :
    :
  int Code;
public:
    Employee *pRight;
    Employee *pLeft;
};
```

Binary Trees

Building a Binary tree using Employee as tree node:

```
class BinaryTree{
   Employee *pParent;
   Employee *insert(Employee *pRoot, Employee *data);
   void inOrder(Employee *pRoot);
   void preOrder(Employee *pRoot);
   void postOrder(Employee *pRoot);
   Employee * deleteT(Employee *pRoot, int key );
  int getHeight(Employee *pRoot);
```

Binary Trees

Building a Binary tree using Employee as tree node:

```
public:
Binary(); {
        pParent = NULL; }
        void insertNode(Employee *data);
        Employee *searchTree(int code);
        void inOrderTraverse();
        void preOrderTraverse();
        void postOrderTraverse();
        void deleteNode(int key );
         int getTreeHeight();
```

Binary Trees

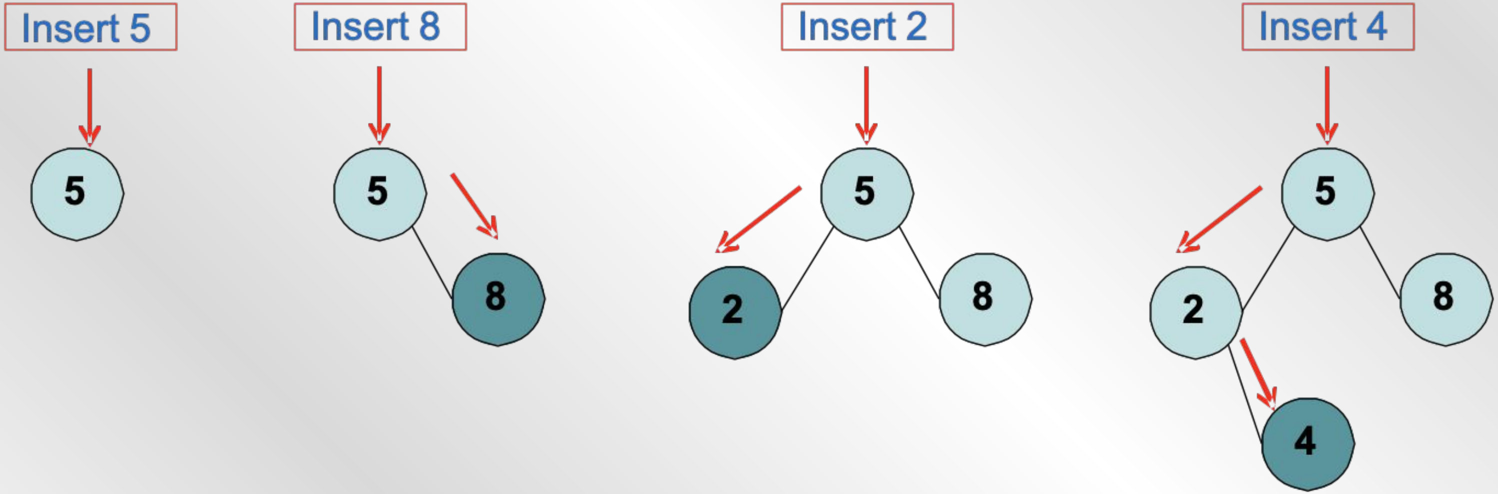
Building a Binary tree [Insert method]:

```
public void BinaryTree:: insertNode ( Employee * data){
pParent = insert(pParent , data); }
private Employee * BinaryTree:: insert ( Employee * pRoot, Employee *data){
// 1. If the tree is empty , return a new single node as a root of the tree
if (pRoot == NULL){
  data>pRight = NULL; data>pLeft = NULL;
   return (data); }
// 2. otherwise , go down to insert it in the right place
else { if (data->getCode() <= pRoot->getCode())
        pRoot->pLeft = insert(pRoot->pLeft,data);
else
      pRoot->pRight= insert(pRoot->pRight,data);
      return(pRoot); // return the unchanged pParent pointer } }
```

Binary Trees

Building a Binary tree [Insert method]

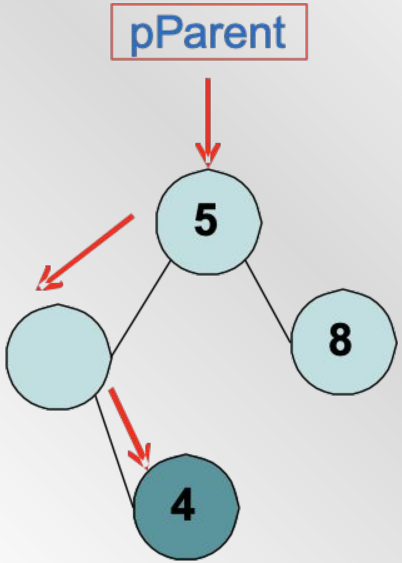
Insert 5,8,2,4



Binary Trees

Building a Binary tree [search method]:

```
public Employee * BinaryTree:: searchTree( int code){
Employee *pRoot;
pRoot = pParent;
while (pRoot && code != pRoot->getCode()) {
    if(code < pRoot ->getCode())
        pRoot = pRoot -> pLeft ;
    else
        pRoot = pRoot -> pRight ; }
  return pRoot; }
```

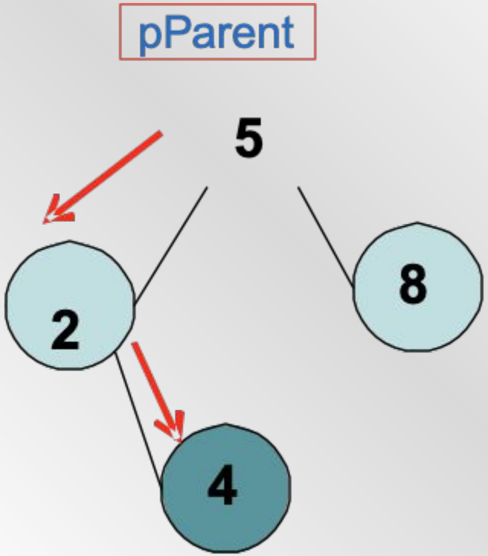


Find 2

Binary Trees

Building a Binary tree [inOrderTraverse method]:

```
public void BinaryTree:: inOrderTraverse (){
        inOrder (pParent);
}
private void BinaryTree:: inOrder ( Employee * pRoot){
  if (pRoot){
      inOrder(pRoot ->pLeft);
      cout<<"Code : " << pRoot ->getCode()<<endl;
      inOrder(pRoot ->pRight);
  }
}
```



Call inOrderTraverse()

Binary Trees

Building a Binary tree [preOrderTraverse method]:

```
public void BinaryTree:: preOrderTraverse (){
      preOrder (pParent);
}
private void BinaryTree:: preOrder( Employee * pRoot){
   if (pRoot){
         cout<<"Code : " << pRoot ->getCode()<<endl;
         preOrder(pRoot ->pLeft);
         preOrder(pRoot ->pRight);
    }
}
```

Call preOrderTraverse()

Binary Trees

Building a Binary tree [postOrderTraverse method]:

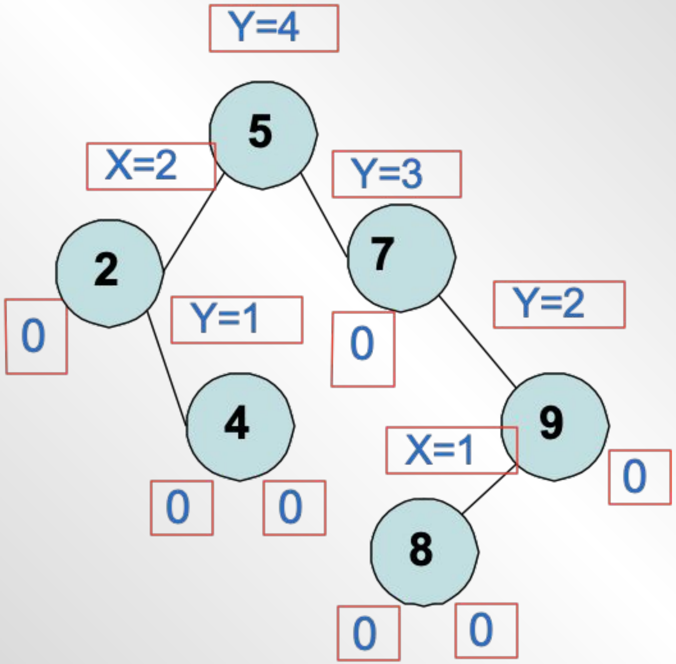
```
public void BinaryTree:: postOrderTraverse (){

            postOrder (pParent);
}
private void BinaryTree:: postOrder ( Employee * pRoot){
    if (pRoot){
        postOrder(pRoot ->pLeft);
        postOrder(pRoot ->pRight);
        cout<<"Code : " << pRoot ->getCode()<<endl;
    }
}
```

Call postOrderTraverse()

Binary Trees

Building a Binary tree [Tree Height method]:



Binary Trees

Building a Binary tree [Delete method]:

There are three cases to consider

- Case 1: delete a leaf node.

  Replace the link to the deleted node by NULL.

- Case 2: delete a node with one subtree.

  The node can be deleted after its parent adjusts a link to bypass the node.

- Case 3: delete a node with two subtree.

  The deleted value must be replaced by an existing value that is either one of the following:

  - The largest value in the deleted node's left subtree

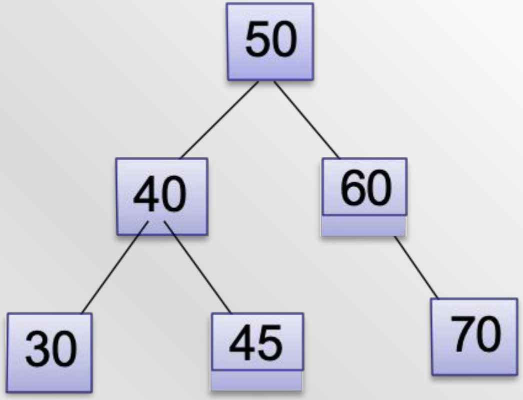
  - The smallest value in the deleted node's right subtree.

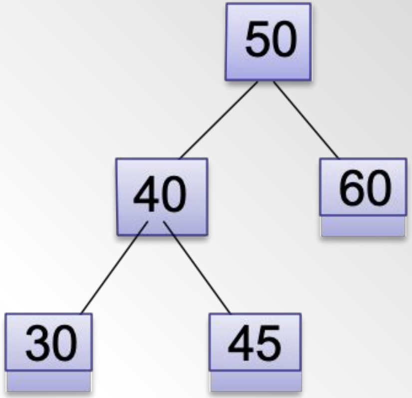
Binary Trees

Building a Binary tree [Delete method]:

Case 1: delete a leaf node [70].

Replace the link to the deleted node by NULL.





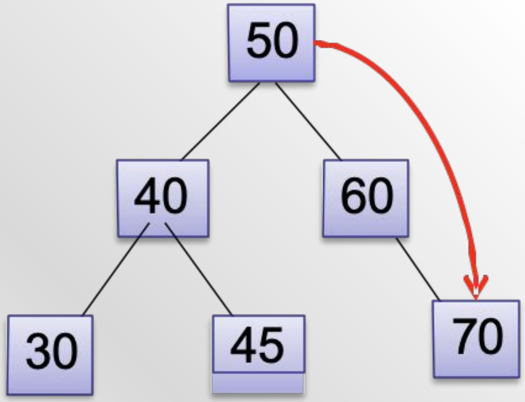
Binary Trees

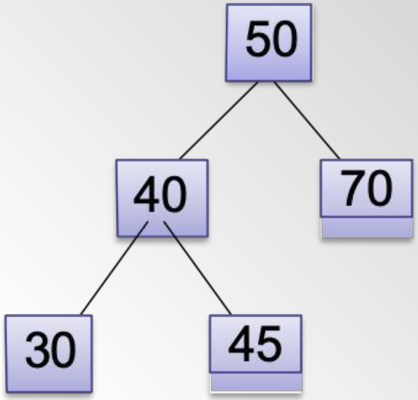
Building a Binary tree [Delete method]:

Case 2: delete a node with one subtree [60].

The node can be deleted after its parent adjusts a link to bypass the node.

.





Binary Trees

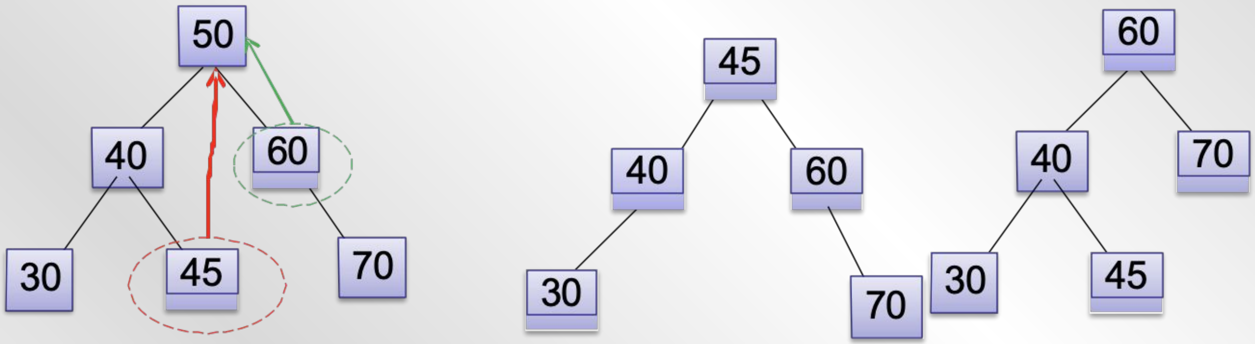
Building a Binary tree [Delete method]:

Case 3: delete a node with two subtree [50].

The deleted value must be replaced for its other childs by an existing value that is either one of the following:

The largest value in the deleted node’s left subtree

The smallest value in the deleted node’s right subtree.



Lab

Exercise

Assignments :

- Implement delete method on Binary Tree.
- Implement and trace getTreeHeight function in the slide on Binary Tree
- Write a code to check if the employee linked list appears as a downward path in a binary tree.

Example:

```
  5
 / \
3   7        5 → 7 → 9
   /
  9
```

- **Bonus:** Implement a function that will count the leaves of a binary tree.

THANKS!