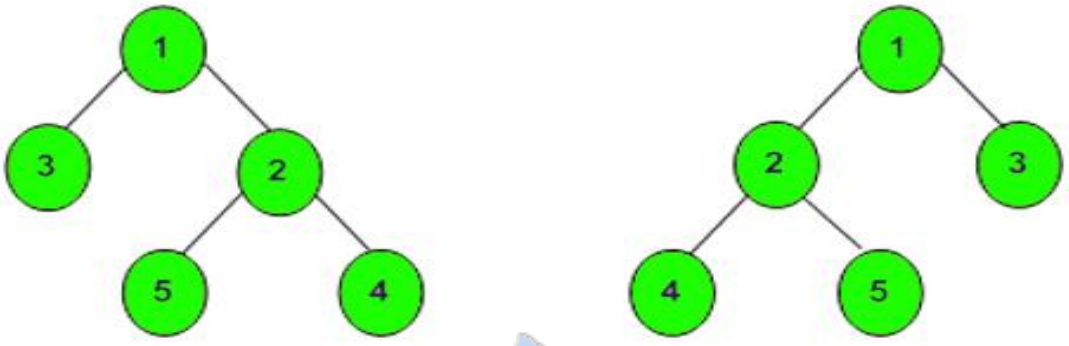
# Trees
# Session 5

# Given a Binary Tree, convert it into its mirror




Mirror Trees

Example 1:

Input:

```
    1
  /  \
  2   3
```

Output: 2 1 3

Explanation: The tree is

```
  1   (mirror)  1
/  \    =>    /  \
3    2       2    3
```

The inorder of mirror is 2 1 3

**Code**

```
1. void mirror(Node* node)
2. {
```

```
    // Your Code Here
    if(node)
    {
        Node *t=node->left;
        node->left=node->right;
        node->right=t;
        mirror(node->left);
          mirror(node->right);
      }
  }
```

# Foldable Binary Tree

Given a binary tree, check if the tree can be folded or not. A tree can be folded if left and right subtrees of the tree are structure wise same. An empty tree is considered as foldable.

Example 1:

```
Input:
    10
   /   \
  7     15
 / \   / \
N   9 11   N
Output:Yes
```

Example 2:

```
Input:
    10
   /   \
  7     15
 / \   / \
5   N 11   N
Output: No
```

**Code**

```
1.bool fold(Node* root1, Node* root2)
```

```
2.{
3.    if(!root1 && !root2)
4.        return true;
5.    if((root1->left==NULL ^ root2->right==NULL)
  || (root2->left==NULL ^ root1->right==NULL))
6.        return false;
7.    return fold(root1->left,root2->right) &&
  fold(root1->right,root2->left);
8.}
9.bool IsFoldable(Node* root)
10. {
11.     // Your code goes here
12.     return fold(root,root);
13. }
```

# Determine if Two Trees are Identical

Given two binary trees, the task is to find if both of them are identical or not.

Example 1:

Input:

```
   1       1
  / \     / \
 2   3   2   3
```

Output: Yes

Explanation: There are two trees bothhaving 3 nodes and 2 edges, both trees are identical having the root as 1, left child of 1 is 2 and right child
of 1 is 3.

Example 2:

Input:

```
   1     1
  / \   / \
 2   3 3   2
```

Output: No

Explanation: There are two trees both having 3 nodes and 2 edges, but both trees are not identical.

**Code**

```
1.bool isIdentical(Node *r1, Node *r2)
```

```
2.{
3.    if(!r1 && !r2)
4.        return 1;
5.    if(r1==NULL ^ r2==NULL)
6.        return 0;
7.    if(r1->data==r2->data)
8.        return isIdentical(r1->left,r2->left) &&
  isIdentical(r1->right,r2->right);
9.    else
10.         return 0;
11. }
```

# Symmetric Tree

Given a Binary Tree. Check whether it is Symmetric or not, i.e. whether the binary tree is a Mirror image of itself or not.

Example 1:

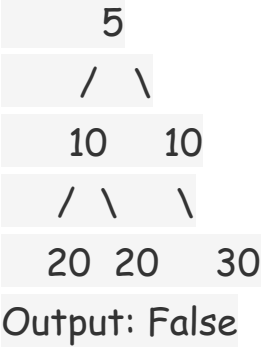
Input:

Output: True

Explanation: Tree is mirror image of itself i.e. tree is symmetric

Example 2:

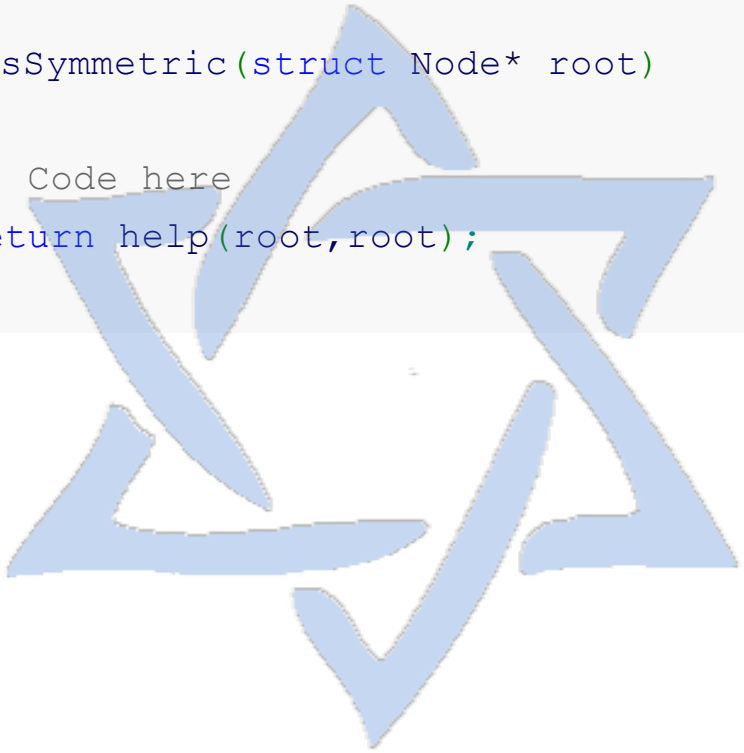
Input:



Output: False

**Code**

```
1.bool help(Node *r1, Node *r2)
2.{
3.    if(!r1 && !r2)
4.        return true;
5.    if(r1 && r2 && r1->data==r2->data)
6.        return help(r1->left,r2->right) &&
  help(r1->right, r2->left);
7.    return false;
8.}
9.bool isSymmetric(struct Node* root)
10. {
11.    // Code here
12.    return help(root,root);
13. }
```



# Check if Tree is Isomorphic

Given two Binary Trees. Check whether they are Isomorphic or not.

Example 1:

```
Input:
 T1   1     T2:  1
    /  \        /  \
   2    3      3   2
  /           /
 4           4
Output: No
```

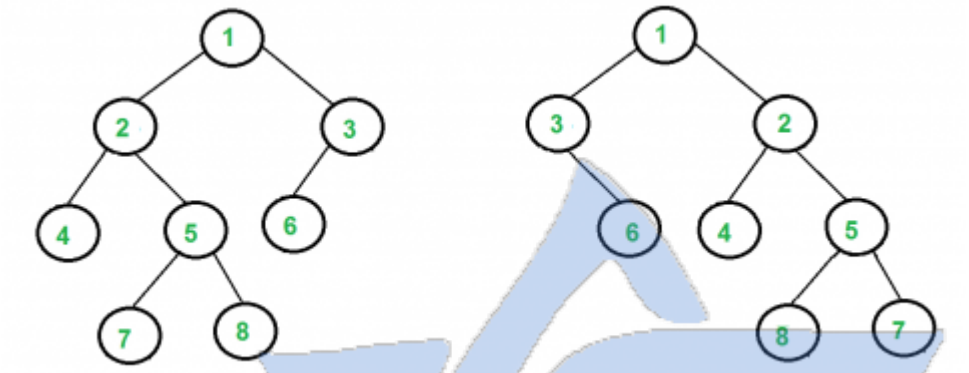
Example 2:

```
Input:
T1   1     T2:   1
   /  \        /   \
  2   3       3    2
 /                  \
 4                   4
Output: Yes
```

Note:

Two trees are called isomorphic if one of them can be obtained from other by a series of flips, i.e. by swapping left and right children of a number of nodes. Any number of nodes at any level can have their children swapped. Two empty trees are isomorphic.

For example, following two trees are isomorphic with following sub-trees flipped: 2 and 3, NULL and 6, 7 and 8.




**Code**

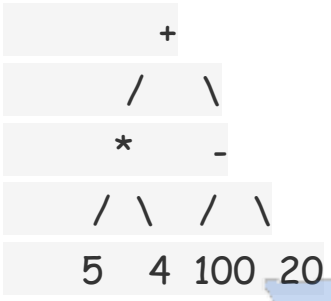
```
1.bool isIsomorphic(Node *root1,Node *root2)
2.{
3.//add code here.
4.    if(root1 && root2)
5.    {
6.        if(root1->data==root2->data)
7.            return
   (isIsomorphic(root1->left,root2->right) &&
   isIsomorphic(root1->right,root2->left)) ||
   (isIsomorphic(root1->left,root2->left) &&
   isIsomorphic(root1->right,root2->right));
8.    }
9.    else if(!root1 && !root2)
10.         return true;
11.     return false;
12. }
```

# Expression Tree

Given a full binary expression tree consisting of basic binary operators (+ , – ,*, /) and some integers, Your task is to evaluate the expression tree.

Example 1:

Input:




Output: 100

Explanation:
((5 * 4) + (100 - 20)) = 100

Example 2:

Input:


Output: -3

Explanation:
4 - 7 = -3

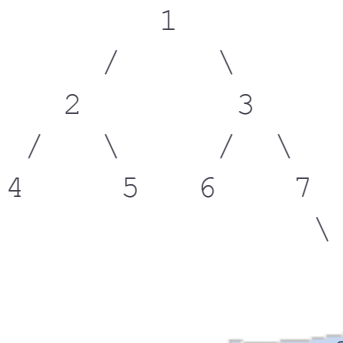
**Code**

```
int evalTree(node* root) {
    // Your code here
    if(root)
    {
        if(!root->left && !root->right)
            return stoi(root->data);
        int r=evalTree(root->right);
        int l=evalTree(root->left);
        if(root->data=="+") return l+r;
          else if(root->data=="-") return l-r;
          else if(root->data=="*") return l*r;
          else return l/r;
    }
    else
        return 0;
}
```

# Print Right View of a Binary Tree

Given a Binary Tree, print Right view of it. Right view of a Binary Tree is set of nodes visible when tree is visited from Right side.

```
Right view of following tree is 1 3 7 8
```




The Right view contains all nodes that are last nodes in their levels. A simple solution is to do level order traversal and print the last node in every level.

The problem can also be solved using simple recursive traversal. We can keep track of level of a node by passing a parameter to all recursive calls. The idea is to keep track of maximum level also. And traverse the tree in a manner that right subtree is visited before left subtree. Whenever we see a node whose level is more than maximum level so far, we print the node because this is the last node in its level (Note that we traverse the right subtree before left subtree). Following is the implementation of this approach.

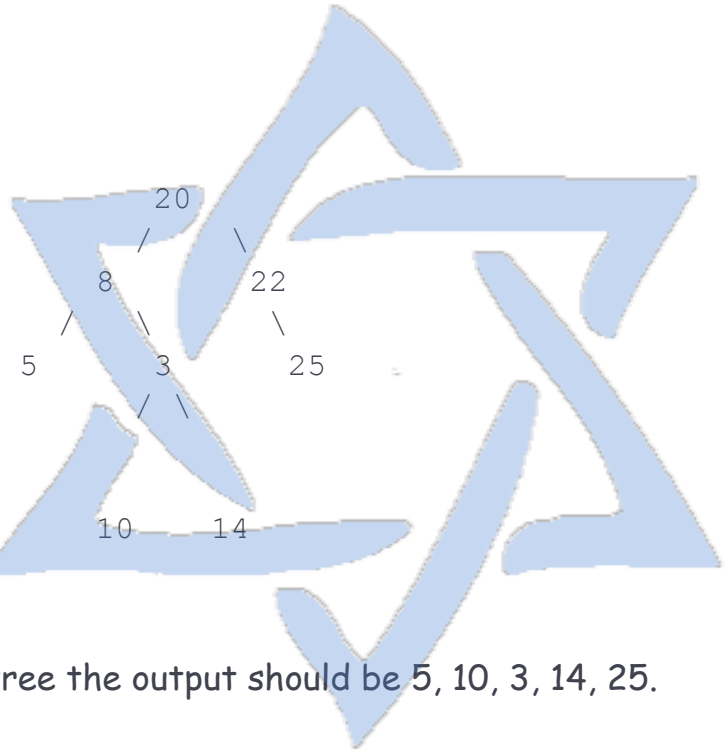
**Code**

```
1.void rightViewUtil(struct Node *root,
2.                   int level, int *max_level)
3.{
4.    // Base Case
5.    if (root == NULL) return;
6.
7.    // If this is the last Node of its level
8.    if (*max_level < level)
9.    {
10.         cout << root->data << "\t";
11.         *max_level = level;
12.     }
13.
14.     // Recur for right subtree first,
15.     // then left subtree
16.     rightViewUtil(root->right, level + 1,
  max_level);
17.     rightViewUtil(root->left, level + 1,
  max_level);
18. }
19.
20. // A wrapper over rightViewUtil()
21. void rightView(struct Node *root)
22. {
23.     int max_level = 0;
24.     rightViewUtil(root, 1, &max_level);
25. }
26.
```

# Bottom View of a Binary Tree

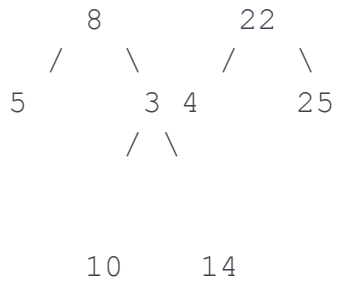
Given a Binary Tree, we need to print the bottom view from left to right. A node x is there in output if x is the bottommost node at its horizontal distance. Horizontal distance of left child of a node x is equal to horizontal distance of x minus 1, and that of right child is horizontal distance of x plus 1.

Examples:




For the above tree the output should be 5, 10, 3, 14, 25.

If there are multiple bottom-most nodes for a horizontal distance from root, then print the later one in level traversal. For example, in the below diagram, 3 and 4 are both the bottom-most nodes at horizontal distance 0, we need to print 4.




For the above tree the output should be 5, 10, 4, 14, 25.

Method 1 – Using Queue

The following are steps to print Bottom View of Binary Tree.

1. We put tree nodes in a queue for the level order traversal.

2. Start with the horizontal distance(hd) 0 of the root node, keep on adding left child to queue along with the horizontal distance as hd-1 and right child as hd+1.

3. Also, use a TreeMap which stores key value pair sorted on key.

4. Every time, we encounter a new horizontal distance or an existing horizontal distance put the node data for the horizontal distance as key. For the first time it will add to the map, next time it will replace the value. This will make sure that the bottom most element for that

horizontal distance is present in the map and if you see the tree from beneath that you will see that element.

Method 2- Using HashMap()

Approach:

Create a map like, map where key is the horizontal distance and value is a pair(a, b) where a is the value of the node and b is the height of the node. Perform a pre-order traversal of the tree. If the current node at a horizontal distance of h is the first we've seen, insert it in the map. Otherwise, compare the node with the existing one in map and if the height of the new node is greater, update in the Map.

Below is the implementation of the above

**Code**:

```
1.void
  bottom(Node*root,map<int,pair<int,int>>&mp,int
  level,int index)
2.{
3.    if(root==NULL)
4.        return;
5.    if (mp.find(index) == mp.end())
6.        mp[index]=make_pair(root->data,level);
7.    else
8.        {
9.            pair<int,int>temp=mp[index];
10.             if(level>=temp.second)
11.             {
12.                 mp[index].second=level;
13.                 mp[index].first=root->data;
14.             }
15.         }
16.    bottom(root->left,mp,level+1,index-1);
```

```
    bottom(root->right,mp,level+1,index+1);
}
// Method that returns the bottom view.
vector <int> bottomView(Node *root)
{
    map<int,pair<int,int>>mp;
    bottom(root,mp,0,0);
    vector<int>v;
    for (auto i = mp.begin(); i != mp.end();
  ++i)
        v.push_back((i->second).first);

    return v;
   // Your Code Here
}
```

# Practice

https://practice.geeksforgeeks.org/problems/left-view-of-binary-tree/1/?category[]=Tree&category[]=Tree&difficulty[]=0&page=2&query=category[]Treedifficulty[]0page2category[]Tree

https://practice.geeksforgeeks.org/problems/vertical-sum/1/?category[]=Tree&category[]=Tree&difficulty[]=0&page=1&query=category[]Treedifficulty[]0page1category[]Tree

https://practice.geeksforgeeks.org/problems/sum-of-right-leaf-nodes/1/?category[]=Tree&category[]=Tree&difficulty[]=0&page=1&query=category[]Treedifficulty[]0page1category[]Tree

Sum of Left Leaf Nodes

https://practice.geeksforgeeks.org/problems/reverse-alternate-levels-of-a-perfect-binary-tree/1/?category[]=Tree&category[]=Tree&difficulty[]=0&page=2&query=category[]Treedifficulty[]0page2category[]Tree

https://practice.geeksforgeeks.org/problems/exchange-the-leaf-nodes/1/?category[]=Tree&category[]=Tree&difficulty[]=0&page=2&query=category[]Treedifficulty[]0page2category[]Tree

https://practice.geeksforgeeks.org/problems/print-all-nodes-that-dont-have-sibling/1/?category[]=Tree&category[]=Tree&difficulty[]=0&page=2&query=category[]Treedifficulty[]0page2category[]Tree

https://practice.geeksforgeeks.org/problems/two-mirror-trees/1/?category[]=Tree&category[]=Tree&difficulty[]=0&page=1&query=category[]Treedifficulty[]0page1category[]Tree

https://practice.geeksforgeeks.org/problems/k-ary-tree1235/1/?category[]=Tree&category[]=Tree&difficulty[]=0&page=2&query=category[]Treedifficulty[]0page2category[]Tree