```
maxDepth('1') = max(maxDepth('2'), maxDepth('3')) + 1
                               = 1 + u1
                                  /    \
                                /        \
                              /            \
                            /                \
                          /                    \
               maxDepth('2') = 1                maxDepth('3') = 0
= max(maxDepth('4'), maxDepth('5')) + 1
= 1 + 0   = 1
                   /    \
                 /        \
               /            \
             /                \
           /                    \
 maxDepth('4') = -1+1 =0     maxDepth('5') = 0
        /       \
 NULL =return -1       NULL
```

```
/* Compute the "maxDepth" of a tree -- the number of
    nodes along the longest path from the root node
    down to the farthest leaf node.*/
int maxDepth(node* node)
{
    if (node == NULL)
        return -1;
    else
    {
        /* compute the depth of each subtree */
```

```
        int lDepth = maxDepth(node->left);

        int rDepth = maxDepth(node->right);


        /* use the larger one */

        if (lDepth > rDepth)

            return(lDepth + 1);

        else return(rDepth + 1);

    }

}

 cout << "Height of tree is " << maxDepth(root);
```

```
tnode *copy(tnode *root)
{
tnode *new_root;
if(root!=NULL)
{
new_root=new tnode;
new_root->data=root->data;
new_root->lchild=copy(root->lchild);
new_root->rchild=copy(root->rchild); } else return NULL; return new_root; }
```

or

```
Tnode* CopyInOrder(Tnode* root)
{
	if(root == NULL)
	{
	return NULL;
	}
	else
	{
		Tnode* temp = new Tnode;
		temp -> data = root -> data;
		temp -> left = copyInOrder(root -> left);
		temp -> right = copyInOrder(root -> right);
		return temp;
	}
}
```

or Node* cloneBinaryTree(Node* root)

```
{
    // base case
    if (root == nullptr) {
        return nullptr;
    }

    // create a new node with the same data as the root node
    Node* root_copy = new Node(root->data);

    // clone the left and right subtree
    root_copy->left = cloneBinaryTree(root->left);
    root_copy->right = cloneBinaryTree(root->right);

    // return cloned root node
    return root_copy;
}
```

Print all leaf nodes of a Binary Tree from left to right

The idea to do this is similar to DFS algorithm. Below is a step by step algorithm to do this:

1. Check if the given node is null. If null, then return from the function.
2. Check if it is a leaf node. If the node is a leaf node, then print its data.
3. If in the above step, the node is not a leaf node then check if the left and right children of node exist. If yes then call the function for left and right child of the node recursively.

```
// function to print leaf
// nodes from left to right
void printLeafNodes(Node *root)
{
    // if node is null, return
    if (!root)
        return;

    // if node is leaf node, print its data
    if (!root->left && !root->right)
    {
        cout << root->data << " ";
        return;
```

```
    }

    // if left child exists, check for leaf
    // recursively
    if (root->left)
       printLeafNodes(root->left);

    // if right child exists, check for leaf
    // recursively
    if (root->right)
       printLeafNodes(root->right);
}
```

```
printLeafNodes(root);
```

or

```
void findLeafNode(Node* root) {
   if (!root)
      return;
   if (!root->left && !root->right) {
```

```
        cout<<root->data<<"\t";
        return;
    }
    if (root->right)
        findLeafNode(root->right);
    if (root->left)
        findLeafNode(root->left);
}
```

Program to count leaf nodes in a binary tree

A node is a leaf node if both left and right child nodes of it are NULL.

Here is an algorithm to get the leaf node count.

```
getLeafCount(node)

1) If node is NULL then return 0.

2) Else If left and right child nodes are NULL return 1.

3) Else recursively calculate leaf count of the tree using below
formula.

    Leaf count of a tree = Leaf count of left subtree +
                                 Leaf count of right subtree
```

```
/* Function to get the count
of leaf nodes in a binary tree*/
unsigned int getLeafCount(struct node* node)
```

```
{
    if(node == NULL)
        return 0;
    if(node->left == NULL && node->right == NULL)
        return 1;
    else
        return getLeafCount(node->left)+
            getLeafCount(node->right);
}
```

```
cout << "Leaf count of the tree is : "<<
                getLeafCount(root) << endl;
```