Tree Traversal

```c
typedef struct Node {
  int item;
  struct node* left;
  struct node* right;
}node;

void inorderTraversal( node* root);
void preorderTraversal( node* root) ;
void postorderTraversal( node* root) ;
node* createNode(int value);
node* insertRight( node* root, int value) ;
node* insertLeft( node* root, int value);
```

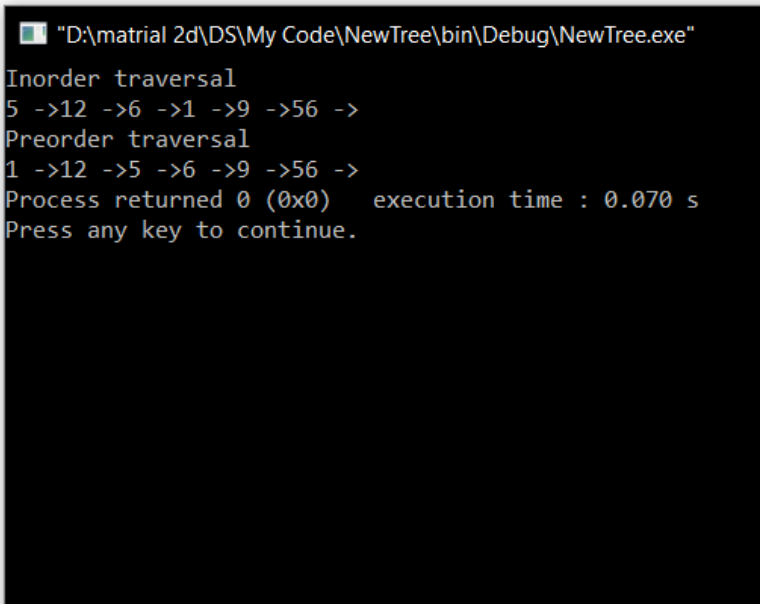
```c
void inorderTraversal( node* root) {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ->", root->item);
  inorderTraversal(root->right);
}
void preorderTraversal( node* root) {
  if (root == NULL) return;
  printf("%d ->", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}
void postorderTraversal( node* root) {
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ->", root->item);
}
node* createNode(int value) {
  node* newNode = malloc(sizeof( node));
  newNode->item = value;
  newNode->left = NULL;
  newNode->right = NULL;
  return newNode;
}
node* insertLeft( node* root, int value) {
  root->left = createNode(value);
  return root->left;
}
// Insert on the right of the node
node* insertRight( node* root, int value) {
  root->right = createNode(value);
  return root->right;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "tree.h"
int main()
{
  node* root = createNode(1);
  insertLeft(root, 12);
  insertRight(root, 9);

  insertLeft(root->left, 5);
  insertRight(root->left, 6);
  insertRight(root->right, 56);

  printf("Inorder traversal \n");
  inorderTraversal(root);

  printf("\nPreorder traversal \n");
  preorderTraversal(root);
    return 0;
}
```



Binary Search (minimum value, Insertion, Search)

```c
pointers. */
struct node* newNode(int data)
{
struct node* node = (struct node*)
                    malloc(sizeof(struct node));
node->data = data;
node->left = NULL;
node->right = NULL;
return(node);
}
struct node* insert(struct node* node, int data)
{
/* 1. If the tree is empty, return a new,
    single node */
if (node == NULL)
    return(newNode(data));
else{
    if (data <= node->data)
        node->left = insert(node->left, data);
    else
        node->right = insert(node->right, data);
    /* return the (unchanged) node pointer */
    return node;
    }
}
int minValue(struct node* node)
{
struct node* current = node;
while (current->left != NULL)
{
    current = current->left;
}
return(current->data);
}
```

```c
bool ifNodeExists( node* node, int key){
if(node==NULL)
        return false;
if (node->item == key)
        return true;

bool res1=ifNodeExists(node->left,key);
if(res1)
    return true;

bool res2=ifNodeExists(node->right,key);
if(res2)
    return true;

}
```

tree.h | main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "tree.h"
int main()
{
  node* root = createNode(12);
  insertLeft(root, 10);

  insertRight(root, 19);

  insertLeft(root->left, 5);
  insertRight(root->left, 11);

  insertLeft(root->right, 18);
  insertRight(root->right, 20);

if(ifNodeExists(root,5))
  printf("yes");
  else
printf("No");


    return 0;
}
```

```
yes
Process returned 0 (0x0)   execution tim
Press any key to continue.
```

Check a binary tree is a full binary tree or not

```
bool isFullBinaryTree(Node* root)
{
    // if tree is empty
    if (!root)
        return true;
    // queue used for level order traversal
    queue<Node*> q;
    // push 'root' to 'q'
    q.push(root);

    // traverse all the nodes of the binary tree
    // level by level until queue is empty
    while (!q.empty()) {
        // get the pointer to 'node' at front
        // of queue
        Node* node = q.front();
        q.pop();
        // if it is a leaf node then continue
        if (node->left == NULL && node->right == NULL)
            continue;
        // if either of the child is not null and the
        // other one is null, then binary tree is not
        // a full binary tee
        if (node->left == NULL || node->right == NULL)
            return false;
        // push left and right childs of 'node'
        // on to the queue 'q'
        q.push(node->left);
        q.push(node->right);
    }
    // binary tree is a full binary tee
    return true;
}
```

```
// Driver program to test above
int main()
{
    Node* root = CreateNode(1);
    root->left = CreateNode(2);
    root->right = CreateNode(3);
    root->left->left = CreateNode(4);
    root->left->right = CreateNode(5);

    if (isFullBinaryTree(root))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

```
Yes
```

Check a tree is a binary tree or not

```
int getMin(root)
{
    BSTNode temp = root
    while(temp.left != NULL)
        temp = temp.left
    return temp.val
}
```

```
int getMax(root)
{
    BSTNode temp = root
    while(temp.right != NULL)
        temp = temp.right
    return temp.val
}
```

```
boolean isBST(BSTNode root)
{
    if (root == NULL)
        return True

    int max_left = getMax(root.left)
    int min_right = getMin(root.right)

    if (max_left > root.val || min_right < root.val)
        return False

    if (isBST(root.left) && isBST(root.right))
        return True
    return False
}
```

Check if Tree is Isomorphic

```
// function to check isomorphic trees
bool isIsomorphic(TreeNode *root1,TreeNode *root2)
{
        if(!root1 && !root2)
        return true;

        if(!root1 || !root2)
        return false;

        if(root1->val!=root2->val)
        return false;

        return ( (isIsomorphic(root1->left,root2->left) &&
                        isIsomorphic(root1->right,root2->right) )||
                        (isIsomorphic(root1->right,root2->left) &&
                        isIsomorphic(root1->left,root2->right)));
}
```

More explanation https://www.includehelp.com/icp/check-if-tree-is-isomorphic.aspx

Program to count leaf nodes in a binary tree

```
/* 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);
}
```

Print the nodes at odd levels of a tree

```
13  void printOddNodes(Node root, bool isOdd)
14 {
15     // If empty tree
16     if (root == null)
17     {
18     return;
19     }
20     // If current node is of odd level
21     if (isOdd == true)
22     {
23     Console.Write(root.data + " ");
24     }
25     // Recur for children with isOdd
26     // switched.
27     printOddNodes(root.left, !isOdd);
28     printOddNodes(root.right, !isOdd);
29 }
30
```

Print all odd nodes of Binary Search Tree

```
// Function to print all odd nodes
void oddNode(Node* root)
{
    if (root != NULL) {
        oddNode(root->left);

        // if node is odd then print it
        if (root->key % 2 != 0)
            printf("%d ", root->key);

        oddNode(root->right);
    }
}
```

More Examples: https://www.geeksforgeeks.org/print-level-order-traversal-line-line/

https://www.geeksforgeeks.org/check-two-bsts-contain-set-elements/

https://www.geeksforgeeks.org/convert-a-given-tree-to-sum-tree/

https://www.geeksforgeeks.org/number-nodes-greater-given-value-n-ary-tree/

https://www.geeksforgeeks.org/write-a-c-program-to-get-count-of-leaf-nodes-in-a-binary-tree/

https://www.geeksforgeeks.org/print-nodes-odd-levels-tree