Implementation of Binary Tree

Aim :

To implement binary search tree using C++.

Algorithm:

1. Start.
2. We create a header file, that contains the operations that could be performed on a binary tree. The header file handles the creation and declaration of nodes using classes.
3. isEmpty() – checks if the tree has any element or not and returns true or false accordingly.
4. insert() – if the root node is empty, then the newly created node is set as the root node. new node's both left and right are always set as null. temp node is used to traverse to the node after which we need to attach the new node. temp node is moved, until null is met. While traversing another node pointer is set such that, it follows temp step behind, so that its easy to compare and decided which side to attach the newly created node.
5. deletenode() – if the root is null, then return null.
   a. If key if less than data of root, then root's left is set as deletion(root->right,key),
   b. Else, root's right is set as deletion(root->left,key).
   c. If both left and right of nodes are NULL, then free root.
   d. If root-> left equals null, then set temp as right of root, and free root and return temp.
   e. If root->right equals null, then set temp as left of root, and free root and return temp.
   f. If none of the above is satisfied then, set the minval(right->root) to temp. copy the data of temp to data of root and set root of right as deletion(root->right,temp->data).
   g. Return root.
6. inorder() – to find the inorder traversal of the tree, first traverse to the left most using recursion until null is encountered. If so then print the data, then the print the parent. Then move to the right sub tree and do the same.
7. preorder() – to find the inorder traversal of the tree, first print the node first, then move to left subtree and repeat the process, once left subtree is done, repeat the same for the right subtree.
8. postorder() – to find the postorder traversal of the tree,move to the first left subtree print data, then the right then the parent, this is done in a recursive way.
9. Counting nodes – if the node's both left and right are NULL, then it is considered a leaf node, else it is considered a internal node.
10. End.

Program:

Creation of header file: *<binary_tree.h>*

```cpp
#include <iostream>
using namespace std;
```

```
class node
{
public:
    int data;
    node *left;
    node *right;
};

bool isEmpty(node *n)
{
    if (n == NULL)
    {
        return true;
    }
    return false;
}

void insert(node **n, int data)
{
    node *new_node = new node();
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    node *temp = *n;
    node *y = NULL;
    if (*n == NULL)
    {
        *n = new_node;
        return;
    }

    while (temp != NULL)
    {
        y = temp;
        if (temp->data < data)
        {
            temp = temp->right;
        }
        else
        {
            temp = temp->left;
        }
    }
    if (data > y->data)
    {
        y->right = new_node;
    }
    else
    {
        y->left = new_node;
    }
}
void tree_search(node **n, int x)
{
    node *temp = *n;
    if (temp->data == x)
    {
        cout << "Data found at root !" << endl;
        return;
    }
    while (temp != NULL && temp->data != x)
    {
        if (temp->data < x)
        {
            temp = temp->right;
        }
        else
        {
            temp = temp->left;
        }
    }
    if (temp == NULL)
    {
        cout << "Element not found !" << endl;
        return;
    }
    cout << "Element found !" << endl;
}


void inorder(node *n)
{
    if (n != NULL)
    {
        inorder(n->left);
        cout << n->data << *" ";
        inorder(n->right);
    }
}

void preorder(node *n)
{
    if (n != NULL)
    {
        cout << n->data << *" ";
        preorder(n->left);
        preorder(n->right);
    }
}

void postorder(node *n)
{
    if (n != NULL)
    {
        postorder(n->left);
        postorder(n->right);
        cout << n->data << *" ";
    }
```

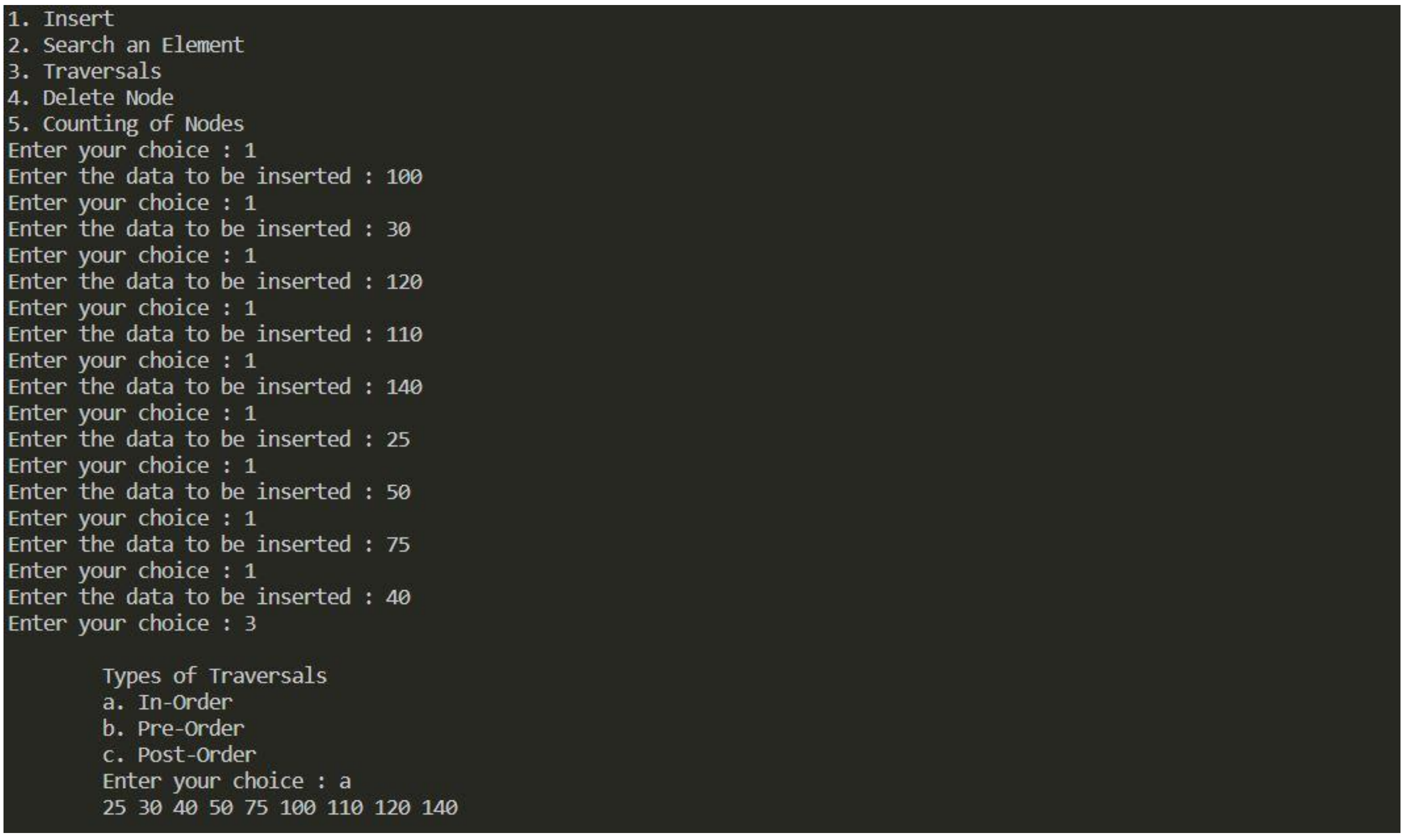
```
}

int internal_count(node *n)
{
    if (n == NULL)
    {
        return 0;
    }
    if (n->left == NULL && n->right == NULL)
    {
        return 0;
    }
    return 1 + internal_count(n->left) + internal_count(n->right);
}

int leaf_count(node *n)
{
    if (n == NULL)
    {
        return 0;
    }
    if (n->left != NULL || n->right != NULL)
    {
        return leaf_count(n->left) + leaf_count(n->right);
    }
    return 1 + leaf_count(n->left) + leaf_count(n->right);
}

node *minval(node *n)
{
    while (n->left != nullptr)
    {
        n = n->left;
    }
    return n;
}
node *deletenode(node *root, int key)
{
    if (root == nullptr)
    {
        return root;
    }
    if (key < root->data)
    {
        root->left = deletenode(root->left, key);
    }
    else if (key > root->data)
    {
        root->right = deletenode(root->right, key);
    }
    else
    {
        if (root->left == nullptr && root->right == nullptr)
        {
```

```cpp
            free(root);
            return nullptr;
        }
        else if (root->left == nullptr)
        {
            node *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == nullptr)
        {
            node *temp = root->left;
            free(root);
            return temp;
        }
        else
        {
            node *temp = minval(root->right);
            root->data = temp->data;
            root->right = deletenode(root->right, temp->data);
        }
    }
    return root;
};
```

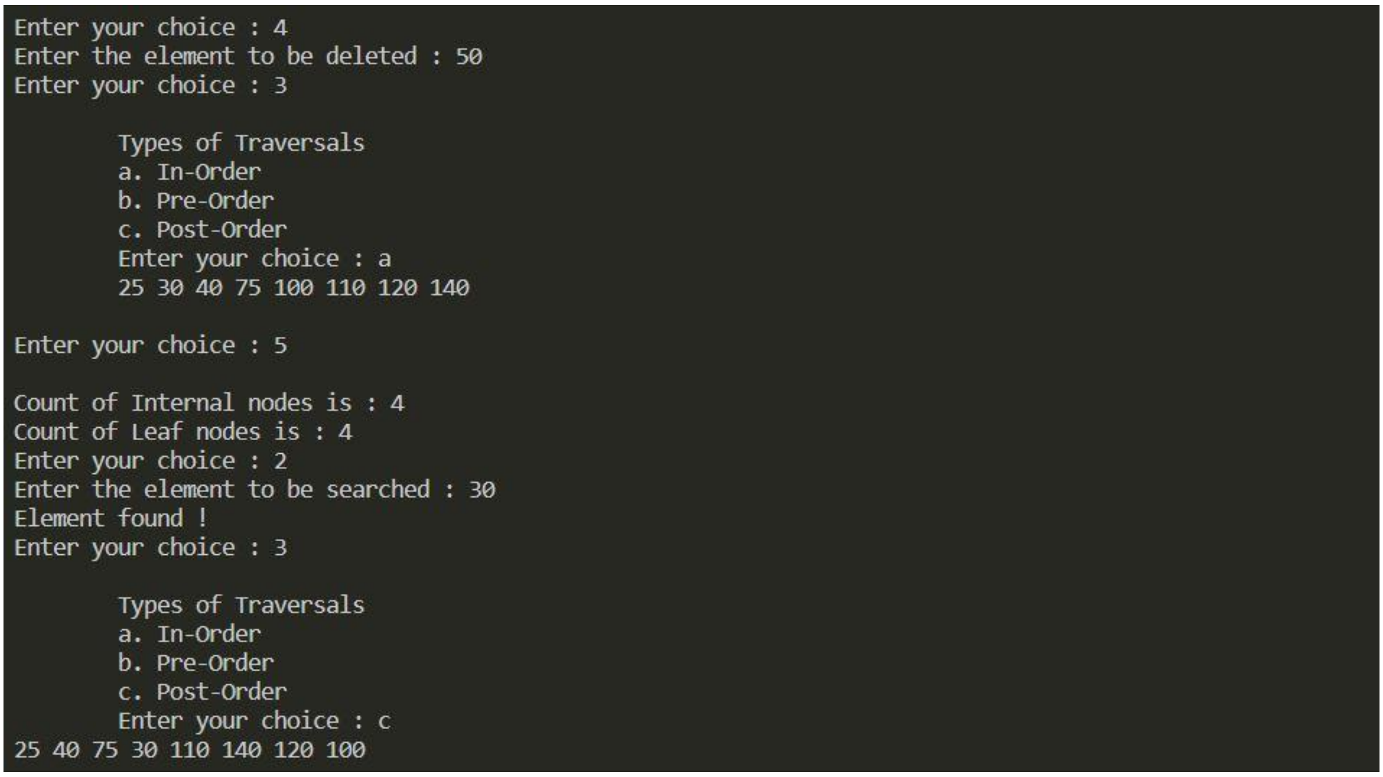
Main Program :

```cpp
#include <iostream>
#include "binary_tree.h"
using namespace std;

int main()
{
    int choice, element;
    node *root = new node();
    root = NULL;
    cout << "1. Insert" << endl;
    cout << "2. Search an Element" << endl;
    cout << "3. Traversals" << endl;
    cout << "4. Delete Node" << endl;
    cout << "5. Counting of Nodes" << endl;
    while (true)
    {
        cout << "Enter your choice : ";
        cin >> choice;
        if (choice == 1)
        {
            cout << "Enter the data to be inserted : ";
            cin >> element;
            insert(&root, element);
        }
        else if (choice == 2)
        {
            int element;
            cout << "Enter the element to be searched : ";
            cin >> element;
            tree_search(&root, element);
        }
        else if (choice == 3)
        {
            if (isEmpty(root))
            {
                cout << "Empty Tree !" << endl;
            }
            else
            {
                char choice;
                cout << "\n\tTypes of Traversals" << endl;
                cout << "\ta. In-Order" << endl;
                cout << "\tb. Pre-Order" << endl;
                cout << "\tc. Post-Order" << endl;
                cout << "\tEnter your choice : ";
                cin >> choice;
                switch (choice)
                {
                case 'a':
                {
                    cout << "\t";
                    inorder(root);
                    cout << endl;
                    break;
                }
                case 'b':
                {
                    preorder(root);
                    cout << endl;
                    break;
                }
                case 'c':
                {
                    postorder(root);
                    cout << endl;
                    break;
                }
                }
                cout << endl;
            }
        }
        else if (choice == 4)
        {
            int element;
            if (isEmpty(root))
            {
                cout << "Empty Tree !" << endl;
            }
            else
            {
                cout << "Enter the element to be deleted : ";
                cin >> element;
                deletenode(root, element);
            }
        }
        else if (choice == 5)
        {
            cout << endl;
            cout << "Count of Internal nodes is : " << internal_count(root) << endl;
            cout << "Count of Leaf nodes is : " << leaf_count(root) << endl;
        }
    }
}
```

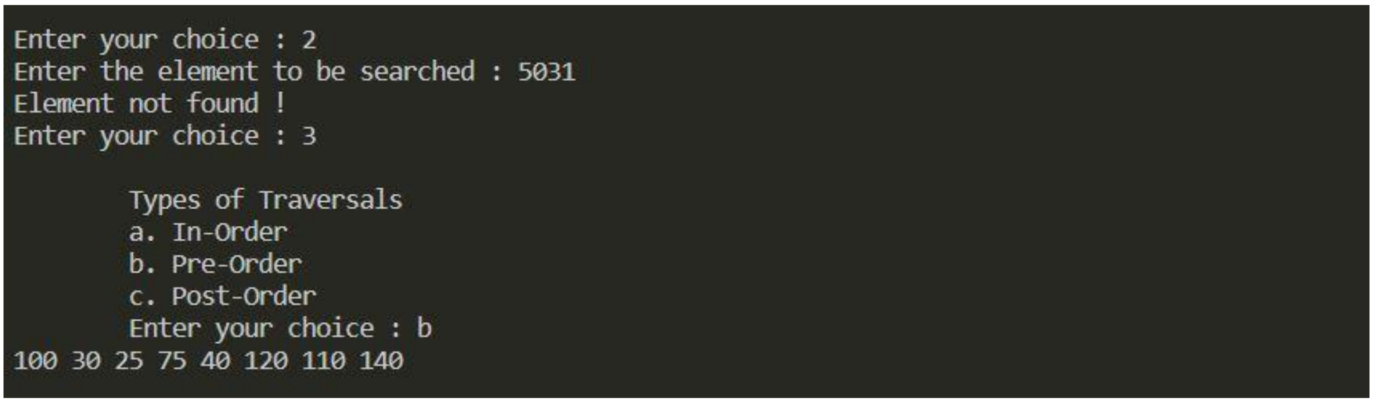
Output:



```
1. Insert
2. Search an Element
3. Traversals
4. Delete Node
5. Counting of Nodes
Enter your choice : 1
Enter the data to be inserted : 100
Enter your choice : 1
Enter the data to be inserted : 30
Enter your choice : 1
Enter the data to be inserted : 120
Enter your choice : 1
Enter the data to be inserted : 110
Enter your choice : 1
Enter the data to be inserted : 140
Enter your choice : 1
Enter the data to be inserted : 25
Enter your choice : 1
Enter the data to be inserted : 50
Enter your choice : 1
Enter the data to be inserted : 75
Enter your choice : 1
Enter the data to be inserted : 40
Enter your choice : 3

        Types of Traversals
        a. In-Order
        b. Pre-Order
        c. Post-Order
        Enter your choice : a
        25 30 40 50 75 100 110 120 140
```



```
Enter your choice : 4
Enter the element to be deleted : 50
Enter your choice : 3

        Types of Traversals
        a. In-Order
        b. Pre-Order
        c. Post-Order
        Enter your choice : a
        25 30 40 75 100 110 120 140

Enter your choice : 5

Count of Internal nodes is : 4
Count of Leaf nodes is : 4
Enter your choice : 2
Enter the element to be searched : 30
Element found !
Enter your choice : 3

        Types of Traversals
        a. In-Order
        b. Pre-Order
        c. Post-Order
        Enter your choice : c
25 40 75 30 110 140 120 100
```



```
Enter your choice : 2
Enter the element to be searched : 5031
Element not found !
Enter your choice : 3

        Types of Traversals
        a. In-Order
        b. Pre-Order
        c. Post-Order
        Enter your choice : b
100 30 25 75 40 120 110 140
```