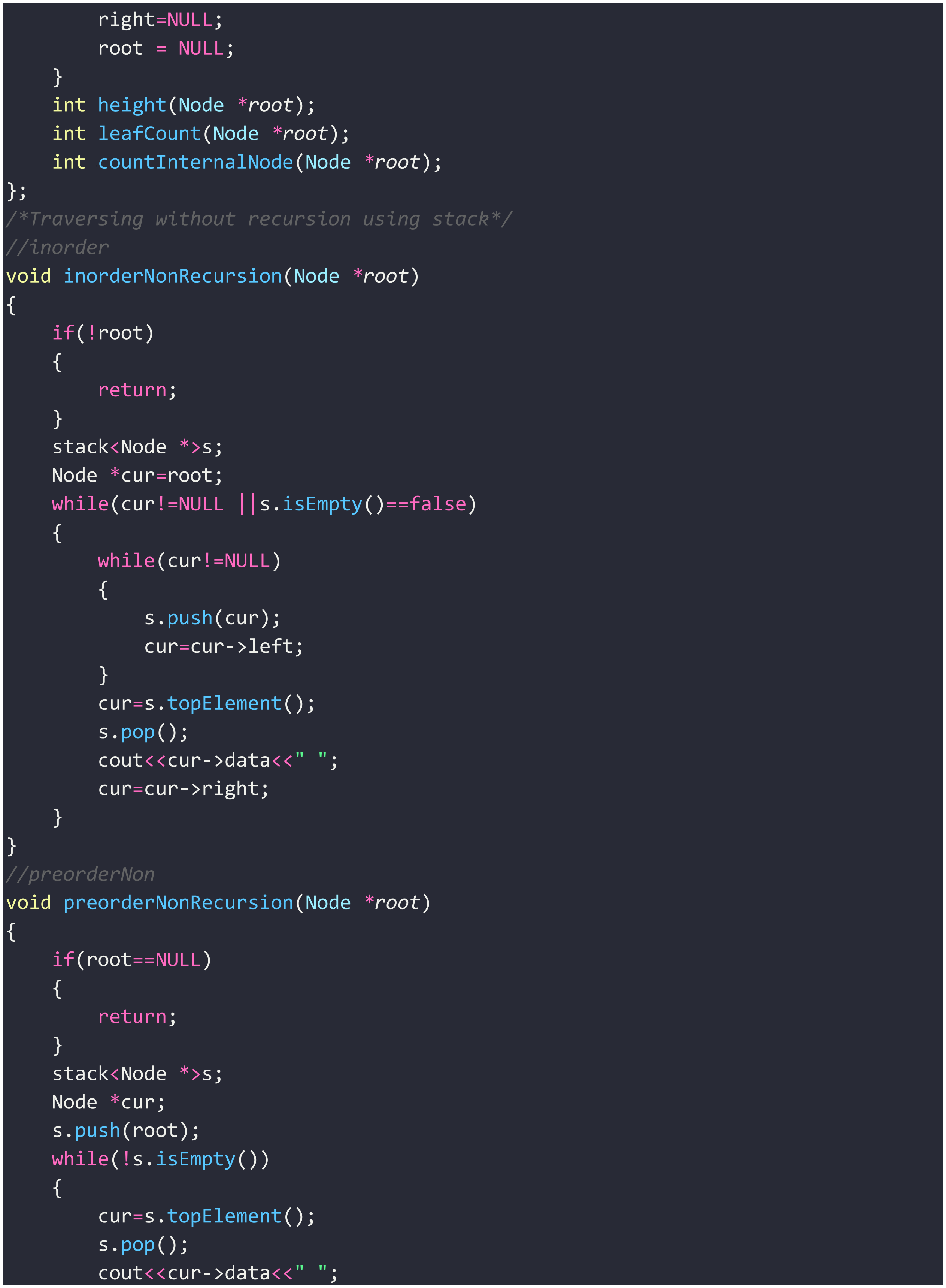
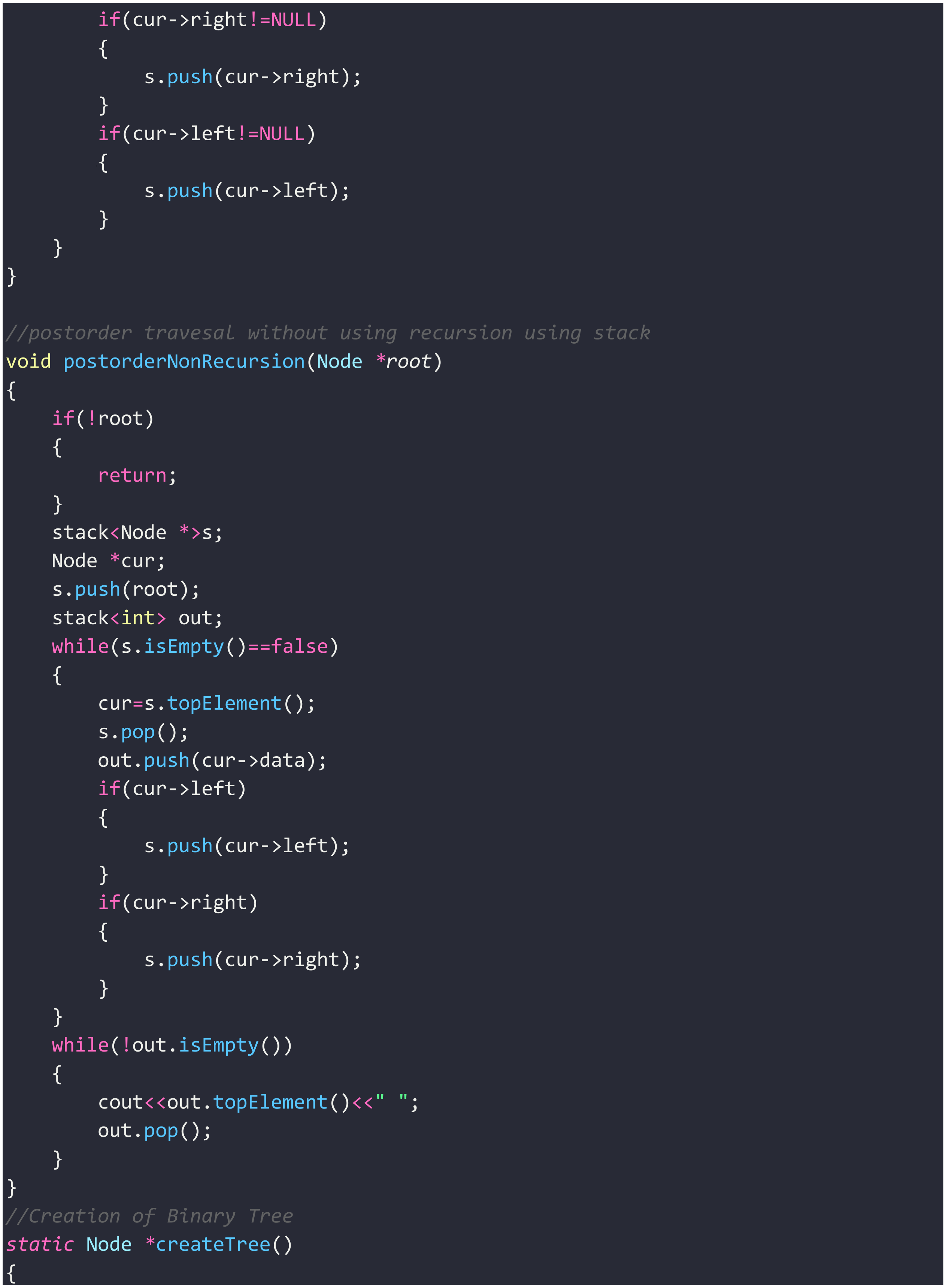
ASSIGNMENT NO – 2

Beginning with an empty binary tree, Construct binary tree by inserting the values in the order given. After constructing a binary tree perform following operations on it- • Perform inorder, preorder and post order traversal • Change a tree so that the roles of the left and right pointers are swapped at every node • Find the height of tree • Copy this tree to another [operator=] • Count number of leaves, number of internal nodes. • Erase all nodes in a binary tree. (Implement both recursive and non-recursive methods)

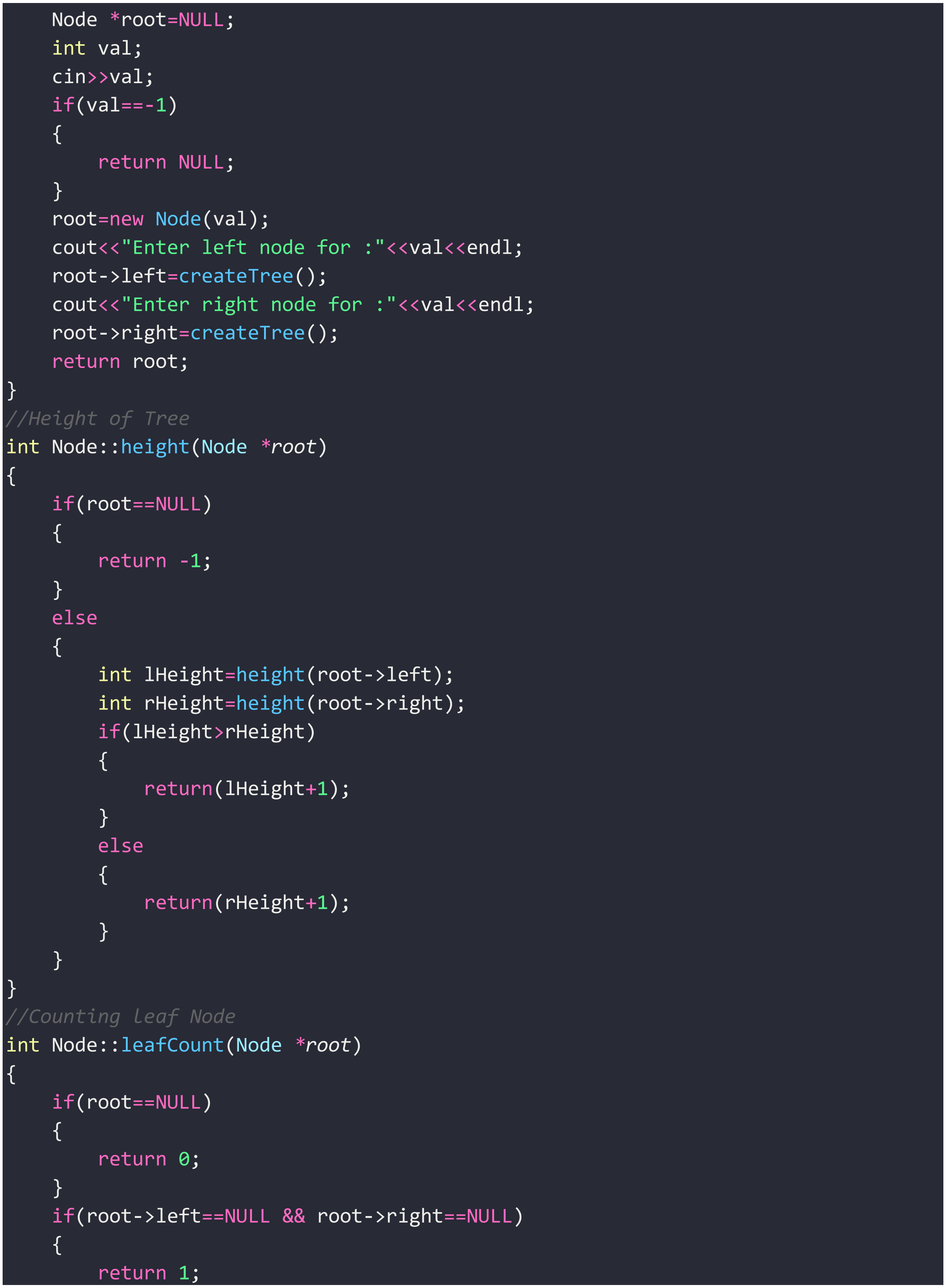
```
#include <stdlib.h>
#include <iostream>
#define SIZE 100
using namespace std;

template <class T> class stack
{
public:
    stack();
    void push(T k);
    T pop();
    T topElement();
    bool isEmpty();
    bool isFull();
private:
    int top;
    T st[SIZE];
};

template <class T> stack<T>::stack()
{
    top=-1;
}

template <class T> void stack<T>::push(T k)
{
    if(isFull())
    {
        cout<<"Stack is Full";
    }
    top=top+1;
    st[top]=k;
}

template <class T> bool stack<T>::isEmpty()
{
    if(top==-1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

template <class T> bool stack<T>::isFull()
{
    if(top==(SIZE-1))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

template <class T> T stack<T>::pop()
{
    T popped_element=st[top];
    top--;
    return popped_element;
}

template <class T> T stack<T>::topElement()
{
    T top_element=st[top];
    return top_element;
}

class Node
{
public:
    Node *left,*right;
    int data;

    Node *root;
    Node(int data)
    {
        this->data=data;
        left=NULL;
```



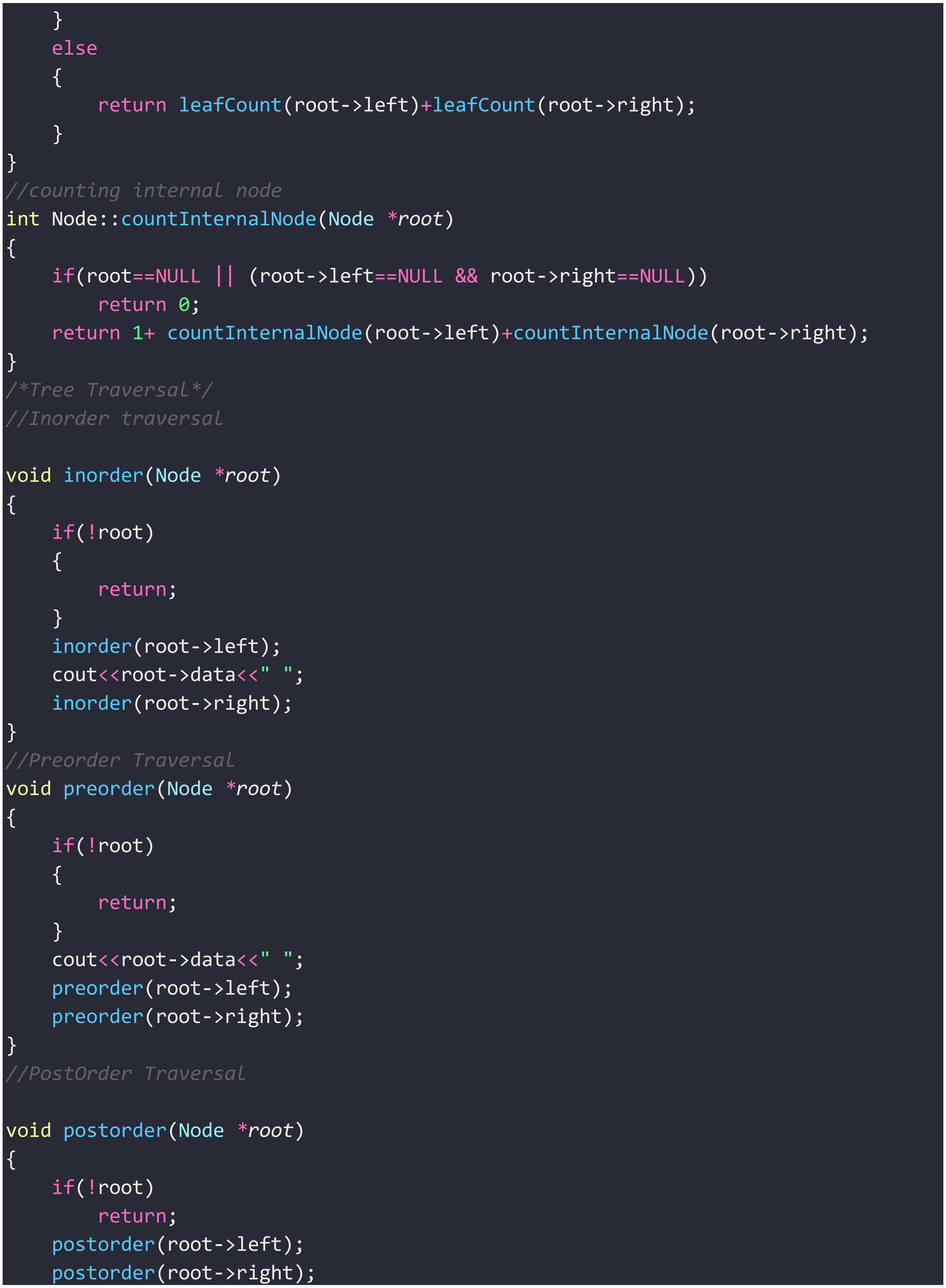
```
        right=NULL;
        root = NULL;
    }
    int height(Node *root);
    int leafCount(Node *root);
    int countInternalNode(Node *root);
};
/*Traversing without recursion using stack*/
//inorder
void inorderNonRecursion(Node *root)
{
    if(!root)
    {
        return;
    }
    stack<Node *>s;
    Node *cur=root;
    while(cur!=NULL ||s.isEmpty()==false)
    {
        while(cur!=NULL)
        {
            s.push(cur);
            cur=cur->left;
        }
        cur=s.topElement();
        s.pop();
        cout<<cur->data<<" ";
        cur=cur->right;
    }
}
//preorderNon
void preorderNonRecursion(Node *root)
{
    if(root==NULL)
    {
        return;
    }
    stack<Node *>s;
    Node *cur;
    s.push(root);
    while(!s.isEmpty())
    {
        cur=s.topElement();
        s.pop();
        cout<<cur->data<<" ";
```



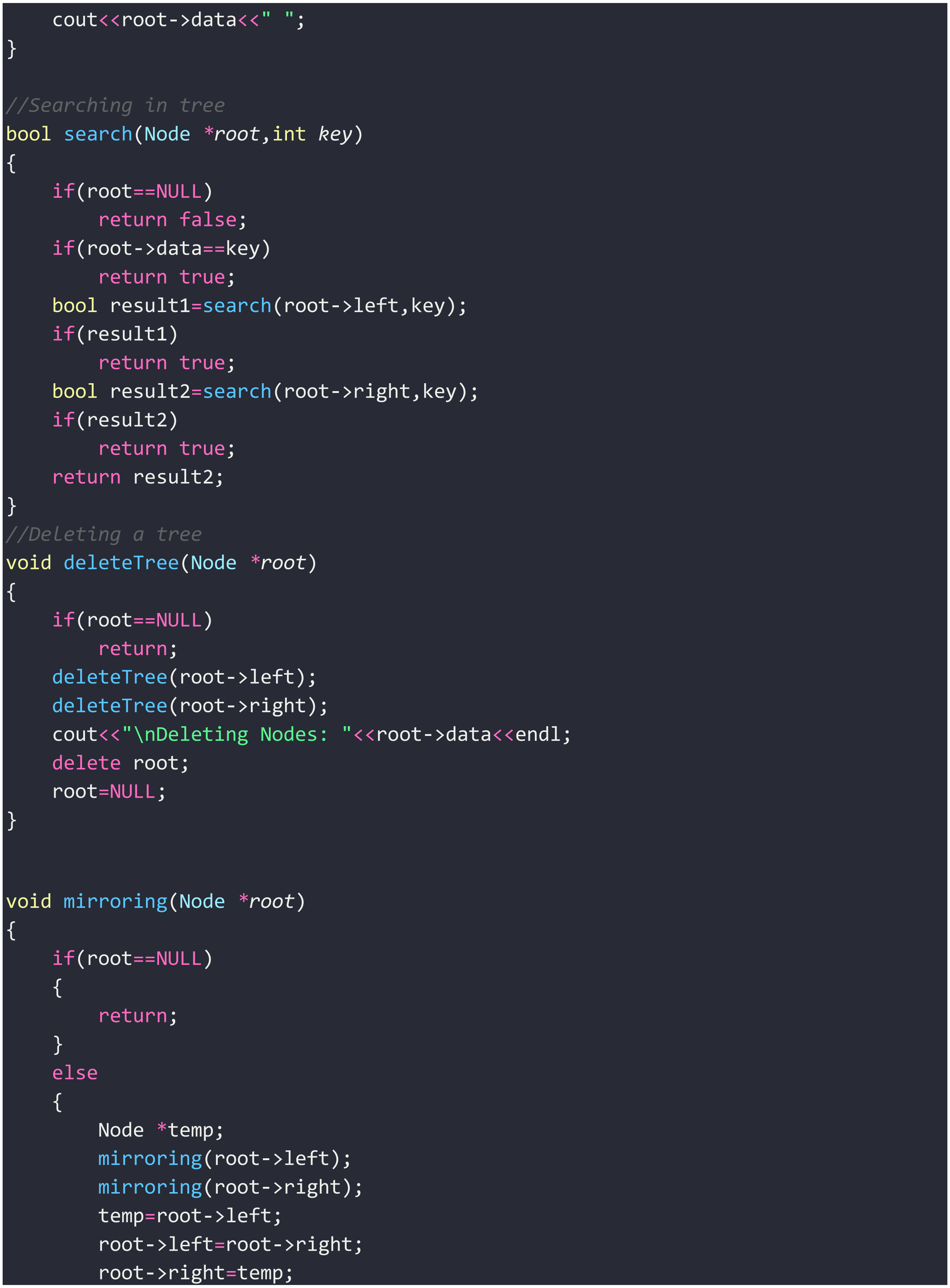
```
        if(cur->right!=NULL)
        {
            s.push(cur->right);
        }
        if(cur->left!=NULL)
        {
            s.push(cur->left);
        }
    }
}

//postorder travesal without using recursion using stack
void postorderNonRecursion(Node *root)
{
    if(!root)
    {
        return;
    }
    stack<Node *>s;
    Node *cur;
    s.push(root);
    stack<int> out;
    while(s.isEmpty()==false)
    {
        cur=s.topElement();
        s.pop();
        out.push(cur->data);
        if(cur->left)
        {
            s.push(cur->left);
        }
        if(cur->right)
        {
            s.push(cur->right);
        }
    }
    while(!out.isEmpty())
    {
        cout<<out.topElement()<<" ";
        out.pop();
    }
}
//Creation of Binary Tree
static Node *createTree()
{
```



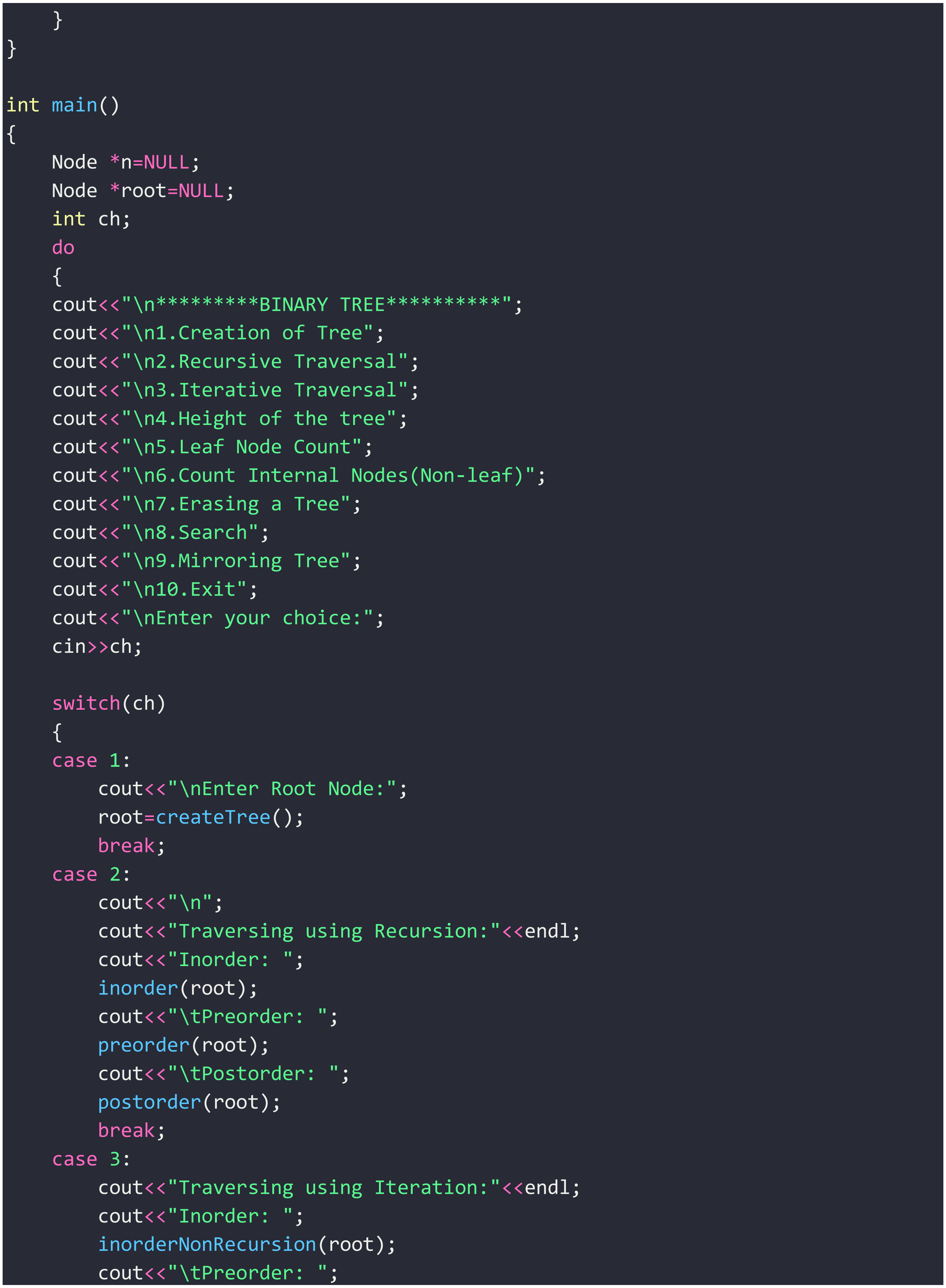
```
    Node *root=NULL;
    int val;
    cin>>val;
    if(val==-1)
    {
        return NULL;
    }
    root=new Node(val);
    cout<<"Enter left node for :"<<val<<endl;
    root->left=createTree();
    cout<<"Enter right node for :"<<val<<endl;
    root->right=createTree();
    return root;
}
//Height of Tree
int Node::height(Node *root)
{
    if(root==NULL)
    {
        return -1;
    }
    else
    {
        int lHeight=height(root->left);
        int rHeight=height(root->right);
        if(lHeight>rHeight)
        {
            return(lHeight+1);
        }
        else
        {
            return(rHeight+1);
        }
    }
}
//Counting leaf Node
int Node::leafCount(Node *root)
{
    if(root==NULL)
    {
        return 0;
    }
    if(root->left==NULL && root->right==NULL)
    {
        return 1;
```



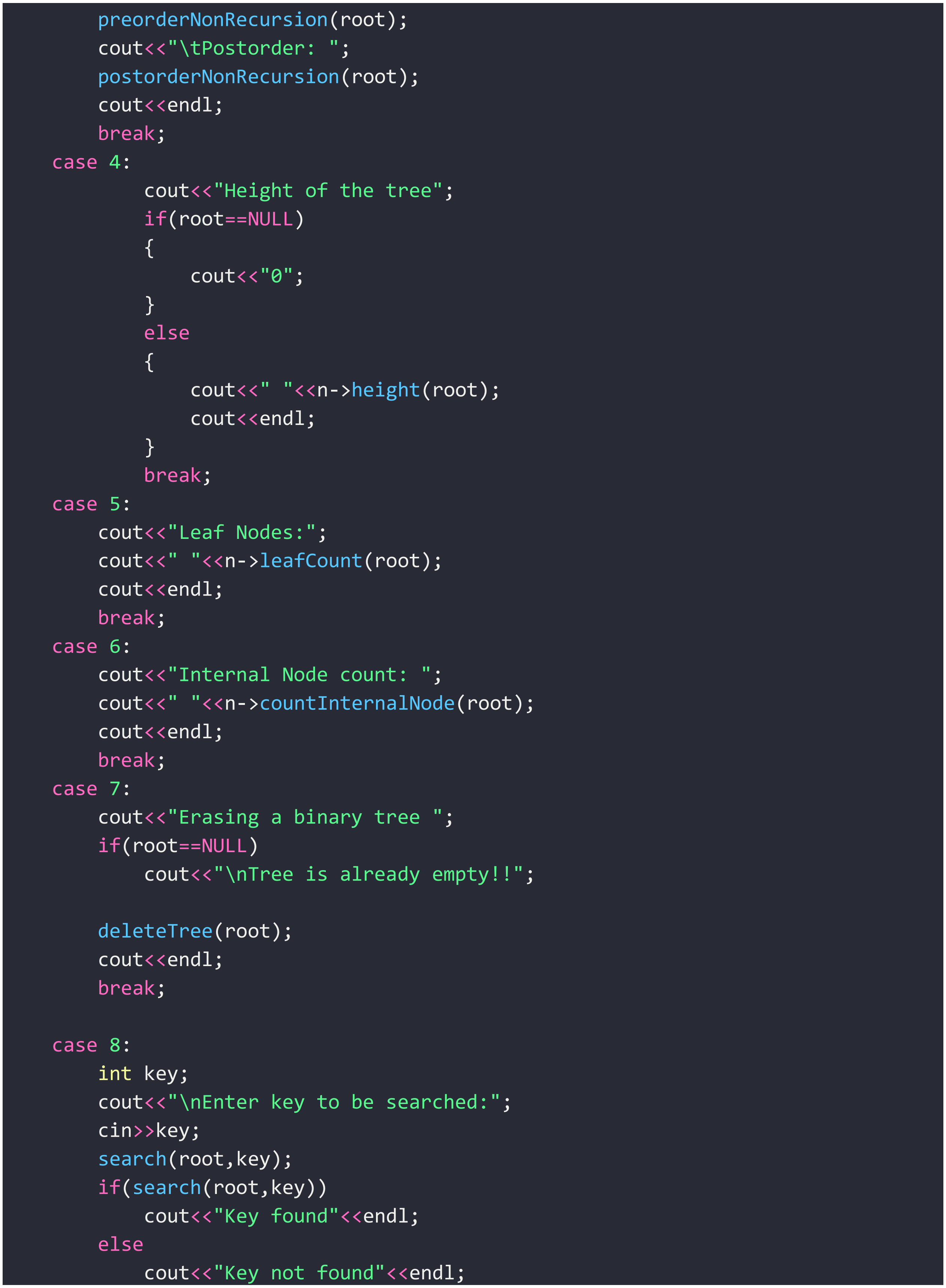
```
    }
    else
    {
        return leafCount(root->left)+leafCount(root->right);
    }
}
//counting internal node
int Node::countInternalNode(Node *root)
{
    if(root==NULL || (root->left==NULL && root->right==NULL))
        return 0;
    return 1+ countInternalNode(root->left)+countInternalNode(root->right);
}
/*Tree Traversal*/
//Inorder traversal

void inorder(Node *root)
{
    if(!root)
    {
        return;
    }
    inorder(root->left);
    cout<<root->data<<" ";
    inorder(root->right);
}
//Preorder Traversal
void preorder(Node *root)
{
    if(!root)
    {
        return;
    }
    cout<<root->data<<" ";
    preorder(root->left);
    preorder(root->right);
}
//PostOrder Traversal

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



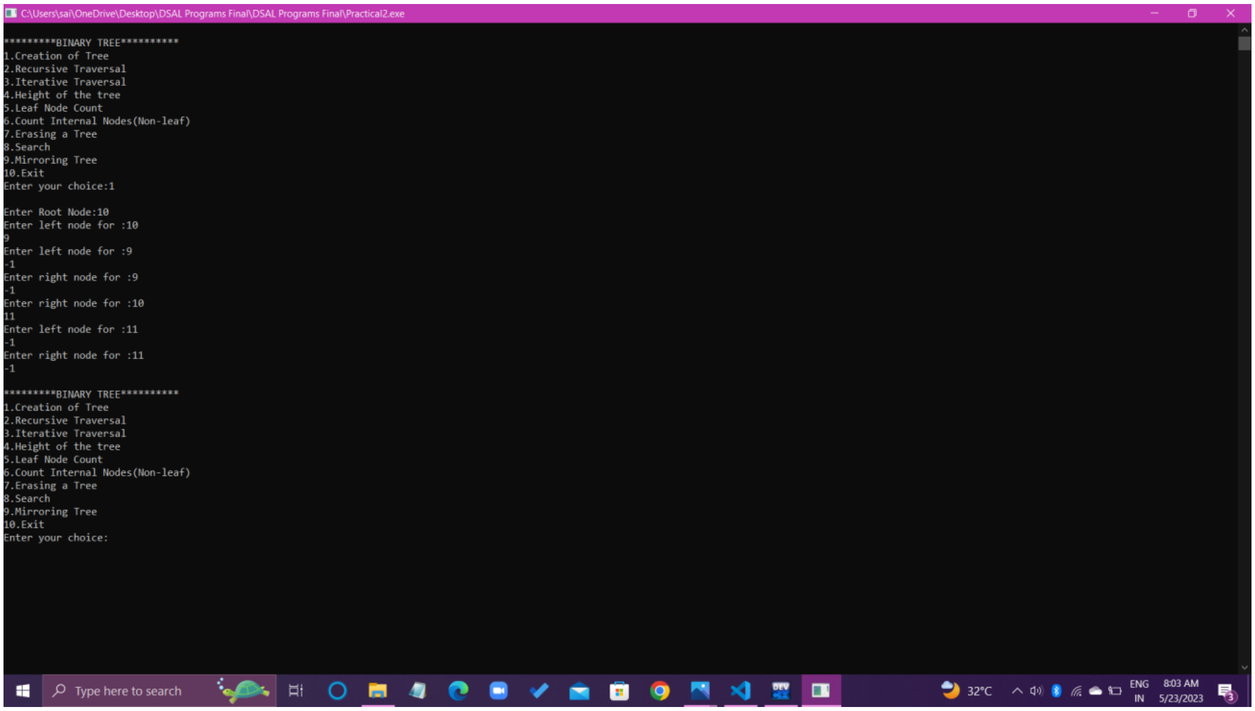
```
    cout<<root->data<<" ";
}

//Searching in tree
bool search(Node *root,int key)
{
    if(root==NULL)
        return false;
    if(root->data==key)
        return true;
    bool result1=search(root->left,key);
    if(result1)
        return true;
    bool result2=search(root->right,key);
    if(result2)
        return true;
    return result2;
}
//Deleting a tree
void deleteTree(Node *root)
{
    if(root==NULL)
        return;
    deleteTree(root->left);
    deleteTree(root->right);
    cout<<"\nDeleting Nodes: "<<root->data<<endl;
    delete root;
    root=NULL;
}


void mirroring(Node *root)
{
    if(root==NULL)
    {
        return;
    }
    else
    {
        Node *temp;
        mirroring(root->left);
        mirroring(root->right);
        temp=root->left;
        root->left=root->right;
        root->right=temp;
```



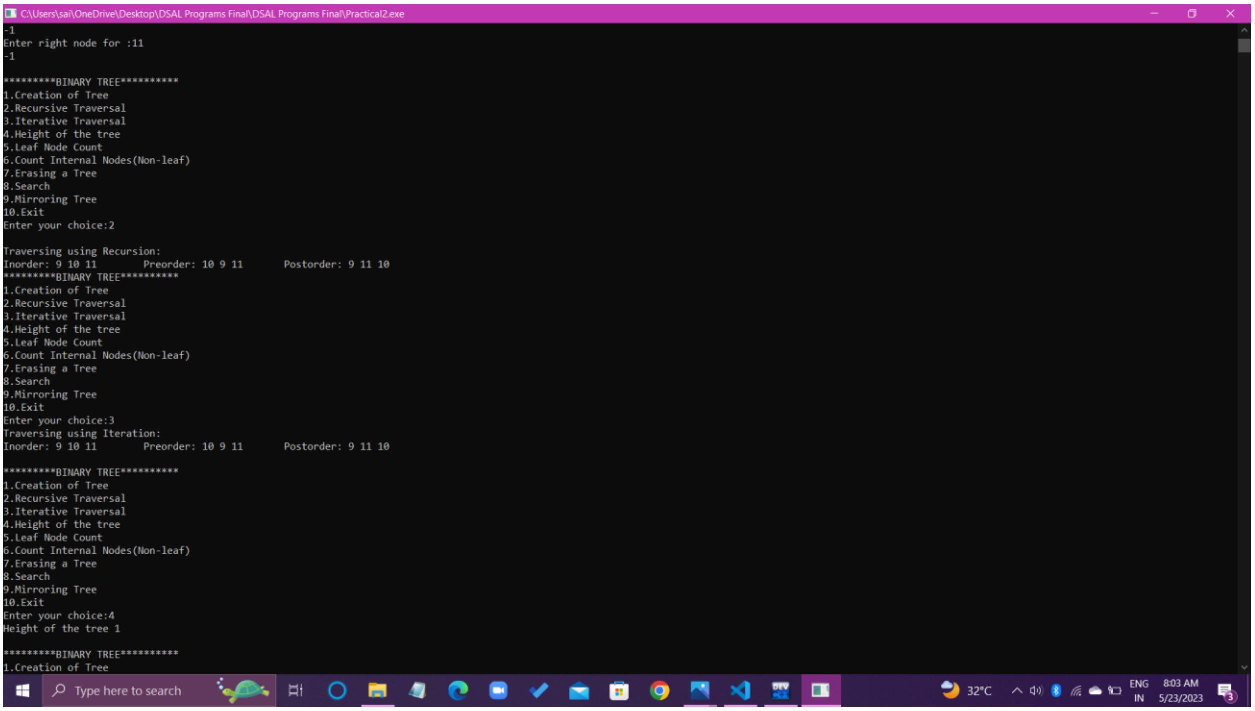
```
    }
}

int main()
{
    Node *n=NULL;
    Node *root=NULL;
    int ch;
    do
    {
    cout<<"\n*********BINARY TREE**********";
    cout<<"\n1.Creation of Tree";
    cout<<"\n2.Recursive Traversal";
    cout<<"\n3.Iterative Traversal";
    cout<<"\n4.Height of the tree";
    cout<<"\n5.Leaf Node Count";
    cout<<"\n6.Count Internal Nodes(Non-leaf)";
    cout<<"\n7.Erasing a Tree";
    cout<<"\n8.Search";
    cout<<"\n9.Mirroring Tree";
    cout<<"\n10.Exit";
    cout<<"\nEnter your choice:";
    cin>>ch;

    switch(ch)
    {
    case 1:
        cout<<"\nEnter Root Node:";
        root=createTree();
        break;
    case 2:
        cout<<"\n";
        cout<<"Traversing using Recursion:"<<endl;
        cout<<"Inorder: ";
        inorder(root);
        cout<<"\tPreorder: ";
        preorder(root);
        cout<<"\tPostorder: ";
        postorder(root);
        break;
    case 3:
        cout<<"Traversing using Iteration:"<<endl;
        cout<<"Inorder: ";
        inorderNonRecursion(root);
        cout<<"\tPreorder: ";
```



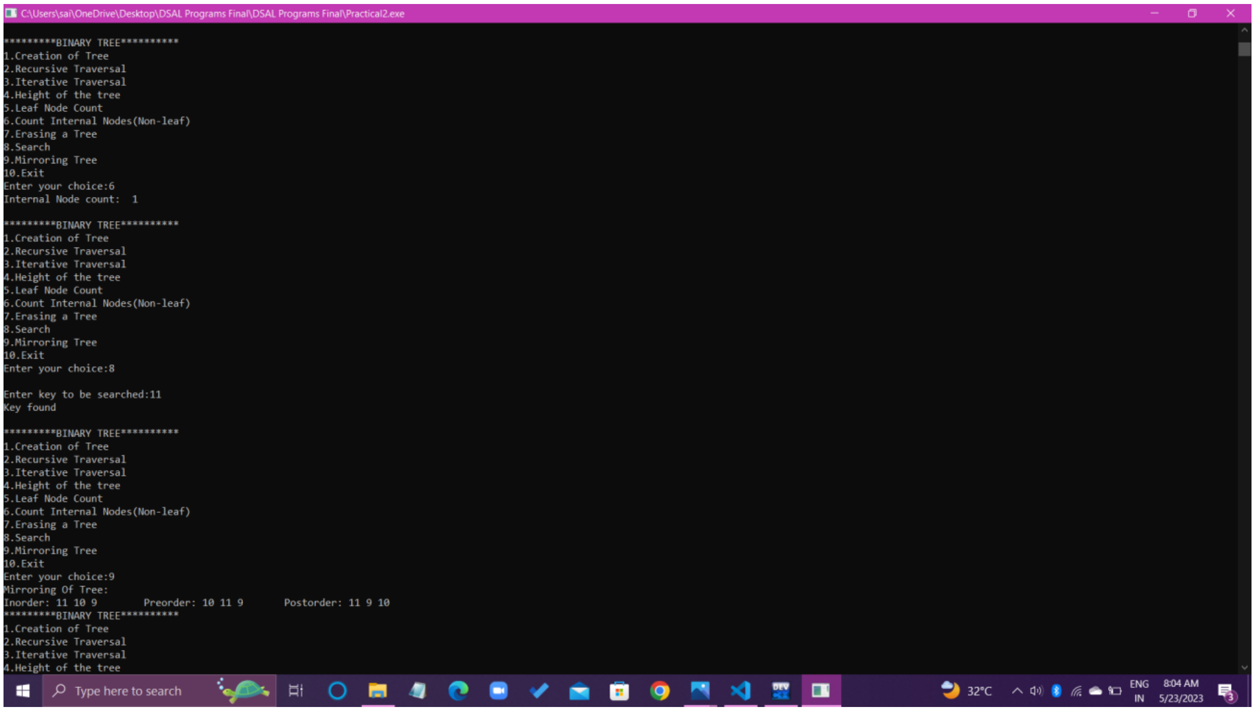
```
        preorderNonRecursion(root);
        cout<<"\tPostorder: ";
        postorderNonRecursion(root);
        cout<<endl;
        break;
    case 4:
            cout<<"Height of the tree";
            if(root==NULL)
            {
                cout<<"0";
            }
            else
            {
                cout<<" "<<n->height(root);
                cout<<endl;
            }
            break;
    case 5:
        cout<<"Leaf Nodes:";
        cout<<" "<<n->leafCount(root);
        cout<<endl;
        break;
    case 6:
        cout<<"Internal Node count: ";
        cout<<" "<<n->countInternalNode(root);
        cout<<endl;
        break;
    case 7:
        cout<<"Erasing a binary tree ";
        if(root==NULL)
            cout<<"\nTree is already empty!!";

        deleteTree(root);
        cout<<endl;
        break;

    case 8:
        int key;
        cout<<"\nEnter key to be searched:";
        cin>>key;
        search(root,key);
        if(search(root,key))
            cout<<"Key found"<<endl;
        else
            cout<<"Key not found"<<endl;
        break;
    case 9:
        cout<<"Mirroring Of Tree: "<<endl;
        mirroring(root);
        cout<<"Inorder: ";
        inorder(root);
        cout<<"\tPreorder: ";
        preorder(root);
        cout<<"\tPostorder: ";
        postorder(root);
        break;
    case 10:
        cout<<"Thank you for using!!!";
        exit(0);
        break;
    default:
        cout<<"Enter correct choice:";
        break;
    }
    }while(ch!=10);

    return 0;
}
```


C:\Users\sai\OneDrive\Desktop\DSAL Programs Final\DSAL Programs Final\Practical2.exe
Type here to search
32°C
ENG
IN
8:03 AM
5/23/2023

```
*********BINARY TREE**********
1.Creation of Tree
2.Recursive Traversal
3.Iterative Traversal
4.Height of the tree
5.Leaf Node Count
6.Count Internal Nodes(Non-leaf)
7.Erasing a Tree
8.Search
9.Mirroring Tree
10.Exit
Enter your choice:1

Enter Root Node:10
Enter left node for :10
9
Enter left node for :9
-1
Enter right node for :9
-1
Enter right node for :10
11
Enter left node for :11
-1
Enter right node for :11
-1

*********BINARY TREE**********
1.Creation of Tree
2.Recursive Traversal
3.Iterative Traversal
4.Height of the tree
5.Leaf Node Count
6.Count Internal Nodes(Non-leaf)
7.Erasing a Tree
8.Search
9.Mirroring Tree
10.Exit
Enter your choice:
```


C:\Users\sai\OneDrive\Desktop\DSAL Programs Final\DSAL Programs Final\Practical2.exe
Type here to search
32°C
ENG
IN
8:03 AM
5/23/2023

```
-1
Enter right node for :11
-1

*********BINARY TREE**********
1.Creation of Tree
2.Recursive Traversal
3.Iterative Traversal
4.Height of the tree
5.Leaf Node Count
6.Count Internal Nodes(Non-leaf)
7.Erasing a Tree
8.Search
9.Mirroring Tree
10.Exit
Enter your choice:2

Traversing using Recursion:
Inorder: 9 10 11        Preorder: 10 9 11       Postorder: 9 11 10
*********BINARY TREE**********
1.Creation of Tree
2.Recursive Traversal
3.Iterative Traversal
4.Height of the tree
5.Leaf Node Count
6.Count Internal Nodes(Non-leaf)
7.Erasing a Tree
8.Search
9.Mirroring Tree
10.Exit
Enter your choice:3
Traversing using Iteration:
Inorder: 9 10 11        Preorder: 10 9 11       Postorder: 9 11 10

*********BINARY TREE**********
1.Creation of Tree
2.Recursive Traversal
3.Iterative Traversal
4.Height of the tree
5.Leaf Node Count
6.Count Internal Nodes(Non-leaf)
7.Erasing a Tree
8.Search
9.Mirroring Tree
10.Exit
Enter your choice:4
Height of the tree 1

*********BINARY TREE**********
1.Creation of Tree
```



C:\Users\sai\OneDrive\Desktop\DSAL Programs Final\DSAL Programs Final\Practical2.exe
Type here to search
32°C
ENG
IN
8:04 AM
5/23/2023

```
*********BINARY TREE**********
1.Creation of Tree
2.Recursive Traversal
3.Iterative Traversal
4.Height of the tree
5.Leaf Node Count
6.Count Internal Nodes(Non-leaf)
7.Erasing a Tree
8.Search
9.Mirroring Tree
10.Exit
Enter your choice:6
Internal Node count:  1

*********BINARY TREE**********
1.Creation of Tree
2.Recursive Traversal
3.Iterative Traversal
4.Height of the tree
5.Leaf Node Count
6.Count Internal Nodes(Non-leaf)
7.Erasing a Tree
8.Search
9.Mirroring Tree
10.Exit
Enter your choice:8

Enter key to be searched:11
Key found

*********BINARY TREE**********
1.Creation of Tree
2.Recursive Traversal
3.Iterative Traversal
4.Height of the tree
5.Leaf Node Count
6.Count Internal Nodes(Non-leaf)
7.Erasing a Tree
8.Search
9.Mirroring Tree
10.Exit
Enter your choice:9
Mirroring Of Tree:
Inorder: 11 10 9        Preorder: 10 11 9       Postorder: 11 9 10
*********BINARY TREE**********
1.Creation of Tree
2.Recursive Traversal
3.Iterative Traversal
4.Height of the tree
```