.vscode\dsa\Binary Tree\binaryTree_Traversal.cpp

```cpp

#include<iostream>
#include<queue>
using namespace std ;

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

    // constructor
    node(int data)
    {
        this->data = data ;
        this->left = NULL ;
        this->right = NULL ;
    }
};

node* buildTree(node* root)
{
    cout<<"enter the data: "<<endl;
    int data ;
    cin>>data ;
    root = new node(data) ;

    if(data == -1)
    {
        return NULL ;
    }

    cout<<"enter data for inserting in left of "<<data<<endl ;
    root->left = buildTree(root->left) ;
    cout<<"enter data for inserting in right of "<<data<<endl ;
    root->right = buildTree(root->right) ;

    return root ;
}

void levelOrderTraversal(node* root)
{
    // breadth first search
    queue<node*> q ;
    q.push(root) ;
    q.push(NULL) ; // it is a separator that differentiates between different levels

    while(!q.empty())
    {
        node* temp = q.front() ;
        q.pop() ;

        if(temp == NULL) // purana level complete traverse ho chuka hai
        {
            cout<<endl ;
            if(!q.empty()) // queue will have some child nodes
            {
                q.push(NULL) ;
            }
        }

        else{
            cout<<temp->data<<" " ;
            if(temp->left != NULL)
            {
            q.push(temp->left) ;
            }
            if(temp->right != NULL)
            {
            q.push(temp->right) ;
            }
        }
    }
}

void inorder(node* root)
{
    // base case
    if(root == NULL)
    {
        return ;
    }
    inorder(root->left) ;
    cout<<root->data<<" " ;
    inorder(root->right) ;
}

void preorder(node* root)
{
    // base case
    if(root == NULL)
    {
        return ;
    }
    cout<<root->data<<" " ;
    preorder(root->left) ;
    preorder(root->right) ;
}

void postorder(node* root)
{
    // base case
    if(root == NULL)
    {
        return ;
```

```cpp
    }
    postorder(root->left) ;
    postorder(root->right) ;
    cout<<root->data<<" " ;
}

void buildFromLevelOrder(node* &root)
{
    queue<node*> q ;
    cout<<"enter data for root - "<<endl;
    int data ;
    cin>>data ;
    root = new node(data) ;
    q.push(root) ;
    while(!q.empty())
    {
        node* temp = q.front() ;
        q.pop() ;

        cout<<"enter left node for : "<<temp->data<<endl;
        int leftData ;
        cin>>leftData ;

        if(leftData != -1)
        {
            temp->left = new node(leftData) ;
            q.push(temp->left) ;
        }

        cout<<"enter right node for : "<<temp->data<<endl;
        int rightData ;
        cin>>rightData ;

        if(rightData != -1)
        {
            temp->right = new node(rightData) ;
            q.push(temp->right) ;
        }
    }
}

int main()
{
    node* root = NULL ;

    // buildFromLevelOrder(root) ;

    root = buildTree(root) ;
    // 1 3 7 -1 -1 11 -1 -1 5 17 -1 -1 -1

    /*
    cout<<"printing the level order traversal -"<<endl;
    levelOrderTraversal(root) ;
    cout<<endl;

    cout<<"inorder traversal is: ";
    inorder(root) ;
    cout<<endl;

    cout<<"preorder traversal is: ";
    preorder(root) ;
    cout<<endl ;

    cout<<"postorder traversal is: ";
    postorder(root) ;
    cout<<endl;
    */

    return 0 ;
}
```