

**.vscode\dsa\Binary Tree\binaryTree\_Traversal.cpp**

```
1
2 #include<iostream>
3 #include<queue>
4 using namespace std ;
5
6 class node{
7     public:
8     int data ;
9     node* left ;
10    node* right ;
11
12    // constructor
13    node(int data)
14    {
15        this->data = data ;
16        this->left = NULL ;
17        this->right = NULL ;
18    }
19 };
20
21 node* buildTree(node* root)
22 {
23     cout<<"enter the data: "<<endl;
24     int data ;
25     cin>>data ;
26     root = new node(data) ;
27
28     if(data == -1)
29     {
30         return NULL ;
31     }
32
33     cout<<"enter data for inserting in left of "<<data<<endl ;
34     root->left = buildTree(root->left) ;
35     cout<<"enter data for inserting in right of "<<data<<endl ;
36     root->right = buildTree(root->right) ;
37
38     return root ;
39 }
40
41 void levelOrderTraversal(node* root)
42 {
43     // breadth first search
44     queue<node*> q ;
45     q.push(root) ;
46     q.push(NULL) ; // it is a separator that differentiates between different levels
47
48     while(!q.empty())
49     {
50         node* temp = q.front() ;
51         q.pop() ;
```

```
52
53     if(temp == NULL) // purana level complete traverse ho chuka hai
54     {
55         cout<<endl ;
56         if(!q.empty()) // queue will have some child nodes
57         {
58             q.push(NULL) ;
59         }
60     }
61
62     else{
63         cout<<temp->data<<" " ;
64         if(temp->left != NULL)
65         {
66             q.push(temp->left) ;
67         }
68         if(temp->right != NULL)
69         {
70             q.push(temp->right) ;
71         }
72     }
73 }
74
75
76 void inorder(node* root)
77 {
78     // base case
79     if(root == NULL)
80     {
81         return ;
82     }
83     inorder(root->left) ;
84     cout<<root->data<<" " ;
85     inorder(root->right) ;
86 }
87
88 void preorder(node* root)
89 {
90     // base case
91     if(root == NULL)
92     {
93         return ;
94     }
95     cout<<root->data<<" " ;
96     preorder(root->left) ;
97     preorder(root->right) ;
98 }
99
100 void postorder(node* root)
101 {
102     // base case
103     if(root == NULL)
104     {
105         return ;
```

```
106     }
107     postorder(root->left) ;
108     postorder(root->right) ;
109     cout<<root->data<<" " ;
110 }
111
112 void buildFromLevelOrder(node* &root)
113 {
114     queue<node*> q ;
115     cout<<"enter data for root - "<<endl;
116     int data ;
117     cin>>data ;
118     root = new node(data) ;
119     q.push(root) ;
120     while(!q.empty())
121     {
122         node* temp = q.front() ;
123         q.pop() ;
124
125         cout<<"enter left node for : "<<temp->data<<endl;
126         int leftData ;
127         cin>>leftData ;
128
129         if(leftData != -1)
130         {
131             temp->left = new node(leftData) ;
132             q.push(temp->left) ;
133         }
134
135         cout<<"enter right node for : "<<temp->data<<endl;
136         int rightData ;
137         cin>>rightData ;
138
139         if(rightData != -1)
140         {
141             temp->right = new node(rightData) ;
142             q.push(temp->right) ;
143         }
144     }
145 }
146
147 int main()
148 {
149     node* root = NULL ;
150
151     // buildFromLevelOrder(root) ;
152
153     root = buildTree(root) ;
154     // 1 3 7 -1 -1 11 -1 -1 5 17 -1 -1 -1
155
156     /*
157     cout<<"printing the level order traversal -"<<endl;
158     levelOrderTraversal(root) ;
159     cout<<endl;
```

```
160
161     cout<<"inorder traversal is: ";
162     inorder(root) ;
163     cout<<endl;
164
165     cout<<"preorder traversal is: ";
166     preorder(root) ;
167     cout<<endl ;
168
169     cout<<"postorder traversal is: ";
170     postorder(root) ;
171     cout<<endl;
172     */
173
174     return 0 ;
175 }
```