

## Binary Tree Code with Recursive Implementation

```
1. #include<iostream>
2. #include<queue>
3. using namespace std;
4. template<class DT>
5. class BNode
6. {
7. public:
8.     BNode()
9.     {
10.         data=0;
11.         leftchild=NULL;
12.         rightchild=NULL;
13.     }
14.     void setLeftChild(BNode<DT>* n)
15.     {
16.         leftchild=n;
17.     }
18.     BNode<DT>* getLeftChild()
19.     {
20.         return leftchild;
21.     }
22.     void setRightChild(BNode<DT>* n)
23.     {
24.         rightchild=n;
25.     }
26.     BNode<DT>* getRightChild()
27.     {
28.         return rightchild;
29.     }
30.     void setData(DT pdate)
31.     {
32.         data=pdate;
33.     }
34.     DT getData()
35.     {
36.         return data;
37.     }
38. private:
39.     DT data;
40.     BNode* leftchild;
41.     BNode* rightchild;
42. };
43. template<class DT>
44. class BinaryTree
45. {
46. public:
47.     //part1: constructor
48.     BinaryTree ()
49.     {
50.         root=NULL;
51.     }
52.     //part 2:
53.     //Build the binary tree from the data given in the array.
54.     //If a node doesn't exist the array element is 0
```

```

55. void BuildTree(DT *Arr, int Size)
56. {
57.     int y,z;
58.     BNode<DT> **temp=new BNode<DT> *[Size];
59.     for(int i=0; i<Size; i++)
60.     {
61.         temp[i]=new BNode<DT>();
62.     }
63.     for(int i=1; i<Size; i++)
64.     {
65.         if(root==NULL)
66.         {
67.             temp[i]->setData(Arr[i]);
68.             root=temp[i];
69.         }
70.         else
71.         {
72.             y=i%2;
73.             if(Arr[i]!=0 && y==0)
74.             {
75.                 z=i/2;
76.                 temp[i]->setData(Arr[i]);
77.                 temp[z]->setLeftChild(temp[i]);
78.             }
79.             else if(Arr[i]!=0 && y==1)
80.             {
81.                 z=i/2;
82.                 temp[i]->setData(Arr[i]);
83.                 temp[z]->setRightChild(temp[i]);
84.             }
85.         }
86.     }
87. }
88.
89. //part3: post order traversal (recursive)
90. //you may call any other function with parameters which might be needed
91. void PostOrderTraversal(BNode<DT>* temp)
92. {
93.     if(temp!=NULL)
94.     {
95.         cout<<temp->getData()<<endl;
96.         PreOrderTraversal(temp->getLeftChild());
97.         PreOrderTraversal(temp->getRightChild());
98.     }
99. }
100. void PostOrder()
101. {
102.     PostOrderTraversal(root);
103. }
104.
105. //part4: pre order traversal (recursive)
106. // you may call any other function with parameters which might be needed
107. void PreOrderTraversal(BNode<DT>* temp)
108. {
109.     if(temp!=NULL)
110.     {
111.         cout<<temp->getData()<<endl;
112.         PreOrderTraversal(temp->getLeftChild());
113.         PreOrderTraversal(temp->getRightChild());
114.     }
115. }
116. void PreOrder()
117. {
118.     PreOrderTraversal(root);
119. }

```

```

120.    //part5: in order traversal (recursive)
121.    // you may call any other function with parameters which might be needed
122.
123.    void InOrderTraversal(BNode<DT>* temp)
124.    {
125.        if(temp!=NULL)
126.        {
127.            InOrderTraversal(temp->getLeftChild());
128.            cout<<temp->getData()<<endl;
129.            InOrderTraversal(temp->getRightChild());
130.        }
131.    }
132.    void InOrder()
133.    {
134.        InOrderTraversal(root);
135.    }
136.
137.    // part6: prints the height of the binary tree, you may pass any parameters needed
138.    int calculateDepth();
139.
140. private:
141.    // you may add any other private members which might be needed by recursive functions
142.    BNode<DT>* root;
143.
144. };
145.
146.
147. int main()
148. {
149.    //creating an object of binary tree
150.    BinaryTree<int> *BT=new BinaryTree<int>();
151.
152.    //array to pass, 0 means no node exists
153.    int Arr[15]={0,1,2,3,4,5,6,7,8,9,10,0,12,13,14};
154.
155.    BT->BuildTree(Arr,15); //building the tree from the array
156.    cout<<"*****"<<endl;
157.
158.    cout<<"Preorder Traversal(Recursive) is: "<<endl;
159.    BT->PreOrder();
160.    cout<<"*****"<<endl;
161.    cout<<"Post order Traversal(Recursive) is: "<<endl;
162.    BT->PostOrder();
163.    cout<<"*****"<<endl;
164.    cout<<"Inorder Traversal(Recursive) is: "<<endl;
165.    BT->InOrder();
166.    cout<<"*****"<<endl;
167.
168.    system("pause");
169.    return 0;
170. }
171.
172.

```

## Output

```
*****
Preorder Traversal(Recursive) is:
1
2
4
8
9
5
10
3
6
12
13
7
14
*****
Post order Traversal(Recursive) is:
1
2
4
8
9
5
10
3
6
12
13
7
14
*****
Inorder Traversal(Recursive) is:
8
4
9
2
10
5
1
12
6
13
3
14
7
*****
```

## Binary Tree with Iterative Implementation

```
1. #ifndef BINARYTREE_H
2. #define BINARYTREE_H
3. template<class DT>
4. class BNode
5. {
6. public:
7.     BNode();
8.     void setLeftChild(BNode<DT>* n);
9.     BNode<DT>* getLeftChild();
10.    void setRightChild(BNode<DT>* n);
11.    BNode<DT>* getRightChild();
12.    void setData(DT pdate);
13.    DT getData();
14. private:
15.    DT data;
16.    BNode* leftchild;
17.    BNode* rightchild;
18. };
19. template<class DT>
20. class BinaryTree
21. {
22. public:
23.     //constructor already done in Lab7, please reuse that code
24.     BinaryTree();
25.
26.     //Build Tree method already done in Lab7, please reuse that code
27.     void BuildTree(DT* Arr, int Size);
28.
29.     //part 1: pre order traversal (iterative)
30.     // If a stack is needed please use the one that comes with C++
31.     void PreOrder();
32.
33.     //part2: in order traversal (iterative)
34.     // If a stack is needed please use the one that comes with C++
35.     void InOrder();
36.
37.     //part3: post order traversal (iterative)
38.     // If a stack is needed please use the one that comes with C++
39.     void PostOrder();
40.
41.     // part4: level order traversal (iterative)
42.     // If a queue is needed please use the one that comes with C++
43.     void LevelOrder();
44.
45.     // part5: calculate and return height of the tree iteratively (iterative)
46.     int calculateHeightItr();
47. private:
48.     BNode<DT>* root;
49. };
50. };
51. #endif
52.
53.
```

54.  
55.

## C++

```
1. #include "BinaryTree.h"
2. #include <iostream>
3. #include <stack>
4. #include <queue>
5. using namespace std;
6. template<class DT>
7. BNode<DT>::BNode()
8. {
9.     leftchild = NULL;
10.    rightchild = NULL;
11. }
12. template<class DT>
13. void BNode<DT>::setLeftChild(BNode<DT>* n)
14. {
15.     leftchild = n;
16. }
17. template<class DT>
18. BNode<DT>* BNode<DT>::getLeftChild()
19. {
20.     return leftchild;
21. }
22. template<class DT>
23. void BNode<DT>::setRightChild(BNode<DT>* n)
24. {
25.     rightchild = n;
26. }
27. template<class DT>
28. BNode<DT>* BNode<DT>::getRightChild()
29. {
30.     return rightchild;
31. }
32. template<class DT>
33. void BNode<DT>::setData(DT pdate)
34. {
35.     data = pdate;
36. }
37. template<class DT>
38. DT BNode<DT>::getData()
39. {
40.     return data;
41. }
42.
43. template<class DT>
44. BinaryTree<DT>::BinaryTree()
45. {
46.     root = NULL;
47. }
48.
```

```

49. template<class DT>
50. void BinaryTree<DT>::BuildTree(DT* Arr, int Size)
51. {
52.     if (Size > 2)
53.     {
54.         BNode<DT>** nodes = new BNode<DT>*[Size];
55.         for (int i = 1; i < Size; ++i)
56.         {
57.             nodes[i] = new BNode<DT>();
58.             if (Arr[i] == 0)
59.             {
60.                 nodes[i] = NULL;
61.             }
62.             else
63.             {
64.                 nodes[i]->setData(Arr[i]);
65.             }
66.             if (i != 1)
67.             {
68.                 if (i % 2 == 0)
69.                 {
70.                     nodes[i / 2]->setLeftChild(nodes[i]);
71.                 }
72.                 else
73.                 {
74.                     nodes[i / 2]->setRightChild(nodes[i]);
75.                 }
76.             }
77.         }
78.         root = nodes[1];
79.     }
80.     else
81.     {
82.         return;
83.     }
84. }
85.
86.
87.
88. template<class DT>
89. void BinaryTree<DT>::PreOrder()
90. {
91.     stack<BNode<DT>*>* s = new stack<BNode<DT>*>();
92.     BNode<DT>* temp = root;
93.     if (temp)
94.     {
95.         s->push(temp);
96.         while (!s->empty())
97.         {
98.             temp = s->top();
99.             s->pop();
100.            cout << temp->getData() << " ";
101.            if (temp->getRightChild())
102.            {
103.                s->push(temp->getRightChild());
104.            }
105.            if (temp->getLeftChild())
106.            {

```

```

107.                                     s->push(temp->getLeftChild());
108.                                     }
109.                                 }
110.                            }
111.                    }
112.
113. template<class DT>
114. void BinaryTree<DT>::PostOrder()
115. {
116.     stack<BNode<DT>*>* s = new stack<BNode<DT>*>();
117.     stack<BNode<DT>*>* t = new stack<BNode<DT>*>();
118.     BNode<DT>* temp = root;
119.
120.     if (temp)
121.     {
122.         s->push(temp);
123.         while (!s->empty())
124.         {
125.             temp = s->top();
126.             s->pop();
127.             t->push(temp);
128.             if (temp->getLeftChild())
129.             {
130.                 s->push(temp->getLeftChild());
131.             }
132.             if (temp->getRightChild())
133.             {
134.                 s->push(temp->getRightChild());
135.             }
136.         }
137.         while (!t->empty())
138.         {
139.             temp = t->top();
140.             t->pop();
141.             cout << temp->getData() << " ";
142.         }
143.     }
144. }
145.
146. template<class DT>
147. void BinaryTree<DT>::InOrder()
148. {
149.     BNode<DT>* temp = root;
150.     stack<BNode<DT>*>* s = new stack<BNode<DT>*>();
151.     if (temp)
152.     {
153.         while (true)
154.         {
155.             if (temp != NULL)
156.             {
157.                 s->push(temp);
158.                 temp = temp->getLeftChild();
159.             }
160.             else
161.             {
162.                 if (!s->empty())
163.                 {
164.                     temp = s->top();
165.                     s->pop();
166.                     cout << temp->getData() << " ";
167.                     temp = temp->getRightChild();
168.                 }
169.                 else

```



```

170.                                     {
171.                                     break;
172.                                     }
173.                                 }
174.                            }
175.                        }
176.    }
177.
178. template<class DT>
179. void BinaryTree<DT>::LevelOrder()
180. {
181.     queue<BNode<DT>*>* q = new queue<BNode<DT>*>();
182.     BNode<DT>* temp = root;
183.
184.     q->push(temp);
185.     while (!q->empty())
186.     {
187.         temp = q->front();
188.         q->pop();
189.         cout << temp->getData() << " ";
190.         if (temp->getLeftChild())
191.         {
192.             q->push(temp->getLeftChild());
193.         }
194.         if (temp->getRightChild())
195.         {
196.             q->push(temp->getRightChild());
197.         }
198.     }
199. }
200.
201.

```

## Main

```

1.
2. #include<iostream>
3. #include<stack>
4. #include<queue>
5. #include "BinaryTree.h"
6. #include "BinaryTree.cpp"
7. using namespace std;
8. int main()
9. {
10.     BinaryTree<int>* BT; //creating an object of binary tree
11.     BT = new BinaryTree<int>();
12.
13.     //array to pass,0 means no node exists
14.     int Arr[15] = { 0,1,2,3,4,5,6,7,8,9,10,0,12,13,14 };
15.
16.     BT->BuildTree(Arr, 15); //building the tree from the array
17.     cout << "*****" << endl;
18.     cout << "Inorder Traversal(Iterative is: " << endl;
19.     BT->InOrder();
20.     cout << endl;
21.     cout << "*****" << endl;
22.     cout << "Preorder Traversal(Iterative) is: " << endl;
23.     BT->PreOrder();
24.     cout << endl;
25.     cout << "*****" << endl;
26.     cout << "Post order Traversal(Iterative) is: " << endl;
27.     BT->PostOrder();
28.     cout << endl;

```

```

29.     cout << "*****" << endl;
30.     cout << "Level order Traversal(Iterative) is: " << endl;
31.     BT->LevelOrder();
32.     cout << endl;
33.     cout << "*****" << endl;
34.
35.     system("pause");
36.     return 0;
37. }
38.

```

## Output

```

*****
Inorder Traversal(Iterative) is:
8 4 9 2 10 5 1 12 6 13 3 14 7
*****
Preorder Traversal(Iterative) is:
1 2 4 8 9 5 10 3 6 12 13 7 14
*****
Post order Traversal(Iterative) is:
8 9 4 10 5 2 12 13 6 14 7 3 1
*****
Level order Traversal(Iterative) is:
1 2 3 4 5 6 7 8 9 10 12 13 14
*****

```

Height of Tree: 4

```

template <class DT>
int BinaryTree<DT>::calculateHeightltr() {

    if (!root) return 0;

    queue<BNode<DT>*> q;

    q.push(root);

    int height = 0;

    while (true) {
        int nodeCount = q.size();
        if (nodeCount == 0) return height;
        height++;

        while (nodeCount > 0) {
            BNode<DT>* temp = q.front();
            q.pop();

            if (temp->getLeftChild()) q.push(temp->getLeftChild());

            if (temp->getRightChild()) q.push(temp->getRightChild());
            nodeCount--;
        }
    }
}

```