Binary Tree Code with Recursive Implementation

```
#include<iostream>
#include<queue>
using namespace std;
template<class DT>
class BNode
{
public:
     BNode()
     {
               data=0;
               leftchild=NULL;
               rightchild=NULL;
     }
     void setLeftChild(BNode<DT>* n)
     {
               leftchild=n;
     }
     BNode<DT>* getLeftChild()
     {
               return leftchild;
     }
     void setRightChild(BNode<DT>* n)
     {
               rightchild=n;
     }
     BNode<DT>* getRightChild()
     {
               return rightchild;
     }
     void setData(DT pdate)
     {
               data=pdate;
     }
     DT getData()
     {
               return data;
     }
private:
     DT data;
     BNode* leftchild;
     BNode* rightchild;
};
template<class DT>
class BinaryTree
{
public:
     //part1: constructor
     BinaryTree ()
     {
               root=NULL;
     }
     //part 2:
     //Build the binary tree from the data given in the array.
     //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

```
#ifndef BINARYTREE_H
#define BINARYTREE_H
template<class DT>
class BNode
{
public:
	BNode();
	void setLeftChild(BNode<DT>* n);
	BNode<DT>* getLeftChild();
	void setRightChild(BNode<DT>* n);
	BNode<DT>* getRightChild();
	void setData(DT pdate);
	DT getData();
private:
	DT data;
	BNode* leftchild;
	BNode* rightchild;
};
template<class DT>
class BinaryTree
{
public:
	//constructor already done in Lab7, please reuse that code
	BinaryTree();

	//Build Tree method already done in Lab7, please reuse that code
	void BuildTree(DT* Arr, int Size);

	//part 1: pre order traversal (iterative)
	// If a stack is needed please use the one that comes with C++
	void PreOrder();

	//part2: in order traversal (iterative)
	// If a stack is needed please use the one that comes with C++
	void InOrder();

	//part3: post order traversal (iterative)
	// If a stack is needed please use the one that comes with C++
	void PostOrder();

	// part4: level order traversal (iterative)
	// If a queue is needed please use the one that comes with C++
	void LevelOrder();

	// part5: calculate and return height of the tree iteratively (iterative)
	int calculateHeightItr();

private:
	BNode<DT>* root;
};
#endif


```

```
54.
55.
```

CPP

```cpp
#include "BinaryTree.h"
#include <iostream>
#include <stack>
#include <queue>
using namespace std;
template<class DT>
BNode<DT>::BNode()
{
     leftchild = NULL;
     rightchild = NULL;
}
template<class DT>
void BNode<DT>::setLeftChild(BNode<DT>* n)
{
     leftchild = n;
}
template<class DT>
BNode<DT>* BNode<DT>::getLeftChild()
{
     return leftchild;
}
template<class DT>
void BNode<DT>::setRightChild(BNode<DT>* n)
{
     rightchild = n;
}
template<class DT>
BNode<DT>* BNode<DT>::getRightChild()
{
     return rightchild;
}
template<class DT>
void BNode<DT>::setData(DT pdate)
{
     data = pdate;
}
template<class DT>
DT BNode<DT>::getData()
{
     return data;
}

template<class DT>
BinaryTree<DT>::BinaryTree()
{
     root = NULL;
}

```

```
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.
63.                         else
64.                         {
65.                                   nodes[i]->setData(Arr[i]);
66.                         }
67.
68.                         if (i != 1)
69.                         {
70.                                   if (i % 2 == 0)
71.                                   {
72.                                             nodes[i / 2]->setLeftChild(nodes[i]);
73.                                   }
74.                                   else
75.                                   {
76.                                             nodes[i / 2]->setRightChild(nodes[i]);
77.                                   }
78.                         }
79.               }
80.               root = nodes[1];
81.     }
82.     else
83.     {
84.               return;
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.                        {
```

```
                                        s->push(temp->getLeftChild());
                                }
                        }
        }
}


template<class DT>
void BinaryTree<DT>::PostOrder()
{
        stack<BNode<DT>*>* s = new stack<BNode<DT>*>();
        stack<BNode<DT>*>* t = new stack<BNode<DT>*>();
        BNode<DT>* temp = root;

        if (temp)
        {
                s->push(temp);
                while (!s->empty())
                {
                        temp = s->top();
                        s->pop();
                        t->push(temp);
                        if (temp->getLeftChild())
                        {
                                s->push(temp->getLeftChild());
                        }
                        if (temp->getRightChild())
                        {
                                s->push(temp->getRightChild());
                        }
                }
                while (!t->empty())
                {
                        temp = t->top();
                        t->pop();
                        cout << temp->getData() << " ";
                }
        }
}

template<class DT>
void BinaryTree<DT>::InOrder()
{
        BNode <DT>* temp = root;
        stack<BNode<DT>*>* s = new stack<BNode<DT>*>();
        if (temp)
        {
                while (true)
                {
                        if (temp != NULL)
                        {
                                s->push(temp);
                                temp = temp->getLeftChild();
                        }
                        else
                        {
                                if (!s->empty())
                                {
                                        temp = s->top();
                                        s->pop();
                                        cout << temp->getData() << " ";
                                        temp = temp->getRightChild();
                                }
                                else
```

```
                                        {
                                                break;
                                        }
                                }
                        }
        }
}

template<class DT>
void BinaryTree<DT>::LevelOrder()
{
        queue<BNode<DT>*>* q = new queue<BNode<DT>*>();
        BNode<DT>* temp = root;

        q->push(temp);
        while (!q->empty())
        {
                temp = q->front();
                q->pop();
                cout << temp->getData() << " ";
                if (temp->getLeftChild())
                {
                        q->push(temp->getLeftChild());
                }
                if (temp->getRightChild())
                {
                        q->push(temp->getRightChild());
                }
        }
}


```

Main

```

#include<iostream>
#include<stack>
#include<queue>
#include "BinaryTree.h"
#include "BinaryTree.cpp"
using namespace std;
int main()
{
        BinaryTree<int>* BT; //creating an object of binary tree
        BT = new BinaryTree<int>();

        //array to pass,0 means no node exists
        int Arr[15] = { 0,1,2,3,4,5,6,7,8,9,10,0,12,13,14 };

        BT->BuildTree(Arr, 15); //building the tree from the array
        cout << "************************************************" << endl;
        cout << "Inorder Traversal(Iterative is: " << endl;
        BT->InOrder();
        cout << endl;
        cout << "************************************************" << endl;
        cout << "Preorder Traversal(Iterative) is: " << endl;
        BT->PreOrder();
        cout << endl;
        cout << "************************************************" << endl;
        cout << "Post order Traversal(Iterative) is: " << endl;
        BT->PostOrder();
        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>::calculateHeightItr() {

        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--;
                    }
                }
      }
```