Binary Tree Code with Recursive Implementation

```cpp
#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
```

```
    void BuildTree(DT *Arr, int Size)
    {
            int y,z;
            BNode<DT> **temp=new BNode<DT> *[Size];
            for(int i=0; i<Size; i++)
            {
                    temp[i]=new BNode<DT>();
            }
            for(int i=1; i<Size; i++)
            {
                    if(root==NULL)
                    {
                            temp[i]->setData(Arr[i]);
                            root=temp[i];
                    }
                    else
                    {
                            y=i%2;
                            if(Arr[i]!=0 && y==0)
                            {
                                    z=i/2;
                                    temp[i]->setData(Arr[i]);
                                    temp[z]->setLeftChild(temp[i]);
                            }
                            else if(Arr[i]!=0 && y==1)
                            {
                                    z=i/2;
                                    temp[i]->setData(Arr[i]);
                                    temp[z]->setRightChild(temp[i]);
                            }
                    }
            }

    }
    //part3: post order traversal (recursive)
    //you may call any other function with parameters which might be needed
    void PostOrderTraversal(BNode<DT>* temp)
    {
            if(temp!=NULL)
            {
                    cout<<temp->getData()<<endl;
                    PreOrderTraversal(temp->getLeftChild());
                    PreOrderTraversal(temp->getRightChild());
            }
    }
    void PostOrder()
    {
            PostOrderTraversal(root);
    }

    //part4: pre order traversal (recursive)
    // you may call any other function with parameters which might be needed
    void PreOrderTraversal(BNode<DT>* temp)
    {
            if(temp!=NULL)
            {
                    cout<<temp->getData()<<endl;
                    PreOrderTraversal(temp->getLeftChild());
                    PreOrderTraversal(temp->getRightChild());
            }
    }
    void PreOrder()
    {
            PreOrderTraversal(root);
    }
```

```
     //part5: in order traversal (recursive)
     // you may call any other function with parameters which might be needed

     void InOrderTraversal(BNode<DT>* temp)
     {
               if(temp!=NULL)
               {
                         InOrderTraversal(temp->getLeftChild());
                         cout<<temp->getData()<<endl;
                         InOrderTraversal(temp->getRightChild());
               }
     }
     void InOrder()
     {
               InOrderTraversal(root);
     }

     // part6: prints the height of the binary tree, you may pass any parameters needed
     int calculateDepth();

private:
// you may add any other private members which might be needed by recursive functions
     BNode<DT>* root;

};


int main()
{
     //creating an object of binary tree
     BinaryTree<int> *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<<"Preorder Traversal(Recursive) is: "<<endl;
     BT->PreOrder();
     cout<<"****************************************************"<<endl;
     cout<<"Post order Traversal(Recursive) is: "<<endl;
     BT->PostOrder();
     cout<<"****************************************************"<<endl;
       cout<<"Inorder Traversal(Recursive) is: "<<endl;
     BT->InOrder();
     cout<<"****************************************************"<<endl;

     system("pause");
     return 0;
}
```

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
```

```


```

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;
}

```

```cpp
template<class DT>
void BinaryTree<DT>::BuildTree(DT* Arr, int Size)
{
    if (Size > 2)
    {
        BNode<DT>** nodes = new BNode<DT>*[Size];
        for (int i = 1; i < Size; ++i)
        {
            nodes[i] = new BNode<DT>();
            if (Arr[i] == 0)
            {
                nodes[i] = NULL;
            }

            else
            {
                nodes[i]->setData(Arr[i]);
            }

            if (i != 1)
            {
                if (i % 2 == 0)
                {
                    nodes[i / 2]->setLeftChild(nodes[i]);
                }
                else
                {
                    nodes[i / 2]->setRightChild(nodes[i]);
                }
            }
        }
        root = nodes[1];
    }
    else
    {
        return;
    }
}

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

```cpp
                {
                    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

```cpp

#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;
    cout << "****************************************************" << endl;
    cout << "Level order Traversal(Iterative) is: " << endl;
    BT->LevelOrder();
    cout << endl;
    cout << "****************************************************" << endl;

    system("pause");
    return 0;
}

```

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--;
                    }
               }
       }
```