TreeNode.h

```
#include <iostream>
using namespace std;
#ifndef TREENODE_H
#define TREENODE_H

class TreeNode {

public:
    friend class BST;
    TreeNode();  //default constructor
    TreeNode(int i, TreeNode* L = 0, TreeNode* R = 0); //explicit value constructor
    int getItem() const; // accessor function
    void setItem(int i);
private:
    int data;
    TreeNode* Lchild;
    TreeNode* Rchild;

};
     inline TreeNode::TreeNode()
        :data(NULL), Lchild(0), Rchild(0)
    {}

      inline TreeNode::TreeNode(int i, TreeNode* L , TreeNode* R )
        : data(i), Lchild(L), Rchild(R)
       {};

      inline int TreeNode::getItem() const
      {
        return data;
       }
      inline void TreeNode::setItem(int i)
      {
          data = i;
      }
#endif
```

BST.h

```
#include "TreeNode.h"
#ifndef BINARY SEARCH TREE
#define BINARY SEARCH TREE

#include <iostream>
using namespace std;


class BST {
public:
	BST();
	void insert(int item);
	void deleteItem(int item);
	void graph(ostream& out);
	bool search( int item);
	void preorder(ostream & out );
	void postorder(ostream & out);
	void inorder(ostream & out);


private:
	TreeNode* root;
	void inorderAux(ostream& out,TreeNode* subtreePtr) ;
	void preorderAux(ostream& out,TreeNode* subtreePtr);
	void postorderAux(ostream& out,TreeNode* subtreePtr);
	void graphAux(ostream& out, int indent, TreeNode* subtreeRoot);
};

#endif
```

BST.cpp

```
#include "BST.h"
#include <iostream>
using namespace std;
BST::BST()
{
    root = NULL;
}
void BST::insert(int item)
{
    TreeNode* locptr = root;
    TreeNode* parent = NULL;        // pointer to parent of current node
       bool found = false;     // indicates if item already in BST
    while (!found && locptr != 0)
    {
        parent = locptr;
        if (item < locptr->data)       // descend left
            locptr = locptr->Lchild;
        else if (locptr->data < item)  // descend right
            locptr = locptr->Rchild;
        else                           // item found
            found = true;
    }
    if (!found)
    {                                 // construct node containing item
        locptr = new TreeNode(item);
        if ( parent == 0)               // empty tree
            root = locptr;
        else if (item < parent->data)  // insert to left of parent
            parent->Lchild = locptr;
        else                           // insert to right of parent
            parent->Rchild = locptr;
    }
    else
        cout << "Item already in the tree\n";
}



void BST::deleteItem(int item)
{
    TreeNode* locptr = root;
    TreeNode* parent = 0;
    bool found = false;
    while (!found && locptr != 0)
    {
        if (item < locptr->data)       // descend left
        {
            parent = locptr;
            locptr = locptr->Lchild;
        }
        else if (locptr->data < item)  // descend right
        {
            parent = locptr;
            locptr = locptr->Rchild;
        }
        else                           // item found
```

```
            found = true;
    }
    if (!found)
    {
        cout << "item isn't found"<<endl;
        return;
    }
    if (locptr->Lchild != 0 && locptr->Rchild != 0)
    {
        TreeNode* succ = locptr->Rchild;
        parent = locptr;
        while (succ->Lchild != 0)
        {
            parent = succ;
            succ = succ->Lchild;
        }
        locptr->data = succ->data;
        locptr = succ;
    }
    TreeNode* sub = locptr->Lchild;
    if (sub == 0)
        sub = locptr->Rchild;
    if (parent == 0)
        root = sub;
    else if (parent->Lchild == locptr)
        parent->Lchild = sub;
    else
        parent->Rchild = sub;
    delete locptr;

}

bool BST::search(int item)
{
    TreeNode* locptr = root;
    bool found = false;
    while (!found && locptr != 0)
    {
        if (item < locptr->data)
            locptr = locptr->Lchild;
        else if (locptr->data < item)
            locptr = locptr->Rchild;
        else
            found = true;
    }
    return found;
}
void BST::preorder(ostream& out)
{
    preorderAux(out, root);
}
void BST::postorder(ostream& out)
{
    postorderAux(out, root);
}
void BST::inorder(ostream& out)
{
    inorderAux(out, root);
```

```
}
void BST::preorderAux(ostream& out, TreeNode* subtreeRoot)
{
    if (subtreeRoot != 0)
    {
        out << subtreeRoot->data << "  ";      // V operation
        preorderAux(out, subtreeRoot->Lchild);    // L operation
        preorderAux(out, subtreeRoot->Rchild);   // R operation
    }
}
void BST::postorderAux(ostream& out, TreeNode* subtreeRoot)
{
    if (subtreeRoot != 0)
    {

        postorderAux(out, subtreeRoot->Lchild);    // L operation
       postorderAux(out, subtreeRoot->Rchild);   // R operation
       out << subtreeRoot->data << "  ";      // V operation

    }
}
void BST::inorderAux(ostream& out,  TreeNode* subtreeRoot)
{
    if (subtreeRoot != 0)
    {
        inorderAux(out, subtreeRoot->Lchild);    // L operation
        out << subtreeRoot->data << "  ";      // V operation
        inorderAux(out, subtreeRoot->Rchild);   // R operation
    }
}
void BST::graph(ostream& out)
{
    graphAux(out, 0, root);
}
#include <iomanip>

void BST::graphAux(ostream& out, int indent,TreeNode* subtreeRoot)
{
    if (subtreeRoot != 0)
    {
        graphAux(out, indent + 8, subtreeRoot->Rchild);
        out << setw(indent) << " " << subtreeRoot->data << endl;
        graphAux(out, indent + 8, subtreeRoot->Lchild);
    }
}
```

Driver.cpp

```
#include <ostream>
#include "BST.h"
#include <iostream>
using namespace std;
int main()
{
    BST tree;
    tree.graph(cout); // create BST
    tree.deleteItem(5); //delete from empty node
    tree.insert(77); //insert numbers
    tree.insert(30);
    tree.insert(90);
    tree.insert(29);
    tree.insert(89);
    tree.insert(156);
    tree.insert(88);
    tree.insert(12);
    tree.insert(3);
    tree.graph(cout); // graph the BST to visualize
    cout << "The number " << (tree.search(158) ? "is" : "is not") << " in the BST\n";
//search non-existing number
    cout << "The number " << (tree.search(12) ? "is" : "is not") << " in the BST\n"; //
search existing number
    tree.inorder(cout); // inorder
    cout << "" << endl;
    tree.preorder(cout); //reorder
    cout << "" << endl;
    tree.postorder(cout); //postorder
    cout << "" << endl;
    tree.deleteItem(1); // delete non-existing node
    tree.deleteItem(12); // delete existing node with 1 child and display the BST
    tree.graph(cout);
    tree.deleteItem(29); // delete node with 2 child and dispay
    tree.graph(cout);
    tree.deleteItem(88); // delete node with 0 child and display
    tree.graph(cout);

}
```