```
#include "library.h"
//Worker
void  inOrder(Book * tree)
{
        if (tree)
        {
                inOrder(tree->getLeftChild());
                tree->print();
                inOrder(tree->getRightChild());
        }
}

void preOrder(Book * tree)
{
        if (tree)
        {
                tree->print();
                preOrder(tree->getLeftChild());
                preOrder(tree->getRightChild());
        }
}

        void postOrder(Book * tree)
{
        if (tree)
        {
                postOrder(tree->getLeftChild());
                postOrder(tree->getRightChild());
                tree->print();
        }
}

//Driver
void Library::printInOrder() const
{
        cout << "\nPrint tree inorder: " << endl;
        inOrder(root);
}

void Library::printPreOrder() const
{
        cout << "\nPrint tree preorder: \n";
        preOrder(root);
}

void Library::printPostOrder() const
{
        cout << "\nPrint tree postorder: \n";
```

```
        postOrder(root);
}

//Implement as a recursive function
int getHeight(Book * tree)
{
        int hl;  //Height of left subtree of tree
        int hr;  //Height of right subtree of tree

        if (tree)
        {
                hl = 1 + getHeight(tree->getLeftChild());
                hr = 1 + getHeight(tree->getRightChild());
                return (hr > hl ? hr : hl);
        }
        else
                return 0;
}

//Driver
int Library::heightIs() const
{
        return getHeight(root);
}

//Return a pointer to a book with maximum isbn starting from "tree"
//                                                      *WORKING*
Book * getMaxBookWorker(Book * tree)
{
        if (tree->getRightChild() == NULL)              // if tree->right is NULL(
no right branch ) then return tree/root bc it is largest
                return tree;
        while (tree->getRightChild())                   // Search for rightmost tree
and set it to tree to be returned
                tree = tree->getRightChild();
        return tree;
}

//Driver                                            *WORKING*
Book * Library::getMaxBook() const
{
        return getMaxBookWorker(root);
}

void deleteWorker( Book * & tree, ItemType item);

//This is a non-recursive function that deletes the node
void deleteNode(Book * & tree)
```

```
{
        ItemType data;
        Book * tempTree;

        Book * t = tree->getLeftChild();
        tempTree = tree;
        if(tree->getLeftChild() == NULL)  //Without left child
        {
                tree = tree->getRightChild(); //move right child to parent
                delete tempTree;
                tempTree = NULL;
        }
        else if(tree->getRightChild() == NULL) //Without right child
        {
                tree = tree->getLeftChild(); //move left child to parent
                delete tempTree;
                tempTree = NULL;
        }
        else  //Have both left and right children
        {
                data = getMaxBookWorker(t)->getInfo();
                deleteWorker(tempTree, data);  // Delete predecessor node.
                tree->setInfo(data);
        }
}

//This recursive function deletes the book with isbn of item.
void deleteWorker( Book * & tree, ItemType item)
{
        if (item < tree->getInfo())
        {
                Book * t = tree->getLeftChild();
                deleteWorker(t, item);            // Look in left subtree.
                tree->setLeftChild(t);
        }
        else if (item > tree->getInfo())
        {
                Book * t = tree->getRightChild();
                deleteWorker(t, item);           // Look in right subtree.
                tree->setRightChild(t);
        }
        else
                deleteNode(tree);                // Node found; call deleteNode.
}

//Driver
void Library::deleteBook(ItemType item)
{
```

```
        deleteWorker(root, item);
}

//Return true if the book with isbn of item on the tree starting from "tree"
void retrieve(Book * tree, ItemType item, bool & found)
{
        if (tree)
        {
                if (item < tree->getInfo())
                {
                        Book * l = tree->getLeftChild();
                        retrieve(l, item, found);
                }
                else if (item > tree->getInfo())
                {
                        Book * r = tree->getRightChild();
                        retrieve(r, item, found);
                }
                else
                {
                        item = tree->getInfo();
                        found = true;
                }
        }
        else
                found = false;
}

//Driver
void Library::retrieveBook(ItemType item, bool & found) const
{
        retrieve(root, item, found);
}

//Count the nodes on the tree                                   *WORKING*
int countNodes(Book * tree)
{
        if (tree == NULL)
                return 0;
        else
                return countNodes(tree->getLeftChild()) +
countNodes(tree->getRightChild()) + 1;
}

//Driver
int Library::lengthIs() const
{
        return countNodes( root );
```

```
}

//Insert a book "b" on the tree                                    *WORKING*
void insert(Book * & tree, Book * b)
{
        if (tree == NULL)
        {
                tree = b;
        }
        else
        {
                if (b->getInfo() > tree->getInfo())
                {
                        Book * tr = tree->getRightChild();
                        insert(tr, b);
                        tree->setRightChild(tr);
                }
                else
                {
                        Book * tl = tree->getLeftChild();
                        insert(tl, b);
                        tree->setLeftChild(tl);
                }
        }
}

//Driver
void Library::insertBook(Book * b)
{
        insert(root, b);
}
```