

```

/*
    St. Cloud State University
    CSCI 301: Introduction to Algorithms and Data Structures
    Reference: Carrano, Fifth edition
    Instructor: Pranava K. Jha
*/
// Implementation file BinaryTree.cpp
// See BinaryTree.h for documentation

#include <iostream>
using namespace std;

#include "BinaryTree.h" // header file

namespace csci301_BinaryTree
{
    BinaryTree::BinaryTree() // default constructor
    {
        root = NULL;
    }

    BinaryTree::BinaryTree(const TreeItemType& rootItem)
    {
        root = new TreeNode(rootItem, NULL, NULL);
    }

    BinaryTree::BinaryTree(const TreeItemType& rootItem,
                           BinaryTree& leftTree,
                           BinaryTree& rightTree)
    {
        root = new TreeNode(rootItem, NULL, NULL);
        attachLeftSubtree(leftTree);
        attachRightSubtree(rightTree);
    } // end constructor

    BinaryTree::BinaryTree(const BinaryTree& tree)
    // copy constructor
    {
        copyTree(tree.root, root);
    }

    BinaryTree::~BinaryTree()
    // destructor
    {
        destroyTree(root);
    }

    bool BinaryTree::isEmpty() const
    {
        return (root == NULL);
    } // end isEmpty

    TreeItemType BinaryTree::getRootData() const
    {
        if (!isEmpty())
            return root->item;
    } // end getRootData

```

```

void BinaryTree::setRootData(const TreeItemType& newItem)
{
    if (!isEmpty())
        root->item = newItem;
    else
        root = new TreeNode(newItem, NULL, NULL);
} // end setRootData

void BinaryTree::attachLeft(const TreeItemType& newItem)
{
    // The existing tree is nonempty and the root has no left child
    if (!isEmpty() && (root->leftChildPtr == NULL))
        root->leftChildPtr = new TreeNode(newItem, NULL, NULL);
} // end attachLeft

void BinaryTree::attachRight(const TreeItemType& newItem)
// The existing tree is nonempty and the root has no right child
{
    if (!isEmpty() && (root->rightChildPtr == NULL))
        root->rightChildPtr = new TreeNode(newItem, NULL, NULL);
} // end attachRight

void BinaryTree::attachLeftSubtree(BinaryTree& leftTree)
// The existing tree is nonempty and the root has no left child
{
    if (!isEmpty() && (root->leftChildPtr == NULL))
    {
        root->leftChildPtr = leftTree.root;
        leftTree.root = NULL;
    }
} // end attachLeftSubtree

void BinaryTree::attachRightSubtree(BinaryTree& rightTree)
// The existing tree is nonempty and the root has no right child
{
    if (!isEmpty() && (root->rightChildPtr == NULL))
    {
        root->rightChildPtr = rightTree.root;
        rightTree.root = NULL;
    } // end if
} // end attachRightSubtree

void BinaryTree::detachLeftSubtree(BinaryTree& leftTree)
{
    if (!isEmpty())
    {
        leftTree = BinaryTree(root->leftChildPtr);
        root->leftChildPtr = NULL;
    } // end if
} // end detachLeftSubtree

```

```

void BinaryTree::detachRightSubtree(BinaryTree& rightTree)
{
    if(!isEmpty())
    {
        rightTree = BinaryTree(root->rightChildPtr);
        root->rightChildPtr = NULL;
    } // end if
} // end detachRightSubtree

BinaryTree BinaryTree::getLeftSubtree() const
{
    TreeNode* subTreePtr;
    if (isEmpty())
        return BinaryTree();
    else
    {
        copyTree(root->leftChildPtr, subTreePtr);
        return BinaryTree(subTreePtr);
    } // end if
} // end getLeftSubtree

BinaryTree BinaryTree::getRightSubtree() const
{
    TreeNode *subTreePtr;
    if (isEmpty())
        return BinaryTree();
    else
    {
        copyTree(root->rightChildPtr, subTreePtr);
        return BinaryTree(subTreePtr);
    } // end if
} // end getRightSubtree

void BinaryTree:: preOrderTraverse()
{
    preOrder(root);
}

void BinaryTree:: inOrderTraverse()
{
    inOrder(root);
}

void BinaryTree:: postOrderTraverse()
{
    postOrder(root);
}

BinaryTree& BinaryTree::operator=(const BinaryTree& rhs)
{
    if (this != &rhs)
    {
        destroyTree(root); // deallocate left-hand side
        copyTree(rhs.root, root); // copy right-hand side
    } // end if
    return *this;
} // end operator=

```

```

BinaryTree::BinaryTree(TreeNode* nodePtr)
// protected constructor
{
    root = nodePtr;
}

void BinaryTree::copyTree(TreeNode *treePtr,
                          TreeNode *& newTreePtr) const
{
// preorder traversal
    if (treePtr != NULL)
    { // copy node
        newTreePtr = new TreeNode(treePtr->item, NULL, NULL);
        copyTree(treePtr->leftChildPtr, newTreePtr->leftChildPtr);
        copyTree(treePtr->rightChildPtr, newTreePtr->rightChildPtr);
    }
    else
        newTreePtr = NULL; // copy empty tree
} // end copyTree

void BinaryTree::destroyTree(TreeNode *& treePtr)
{
// postorder traversal
    if (treePtr != NULL)
    {
        destroyTree(treePtr->leftChildPtr);
        destroyTree(treePtr->rightChildPtr);
        delete treePtr;
        treePtr = NULL;
    } // end if
} // end destroyTree

TreeNode* BinaryTree::rootPtr() const
{
    return root;
} // end rootPtr

void BinaryTree::setRootPtr(TreeNode *newRoot)
{
    root = newRoot;
} // end setRoot

void BinaryTree::getChildPtrs(TreeNode *nodePtr,
                              TreeNode *& leftPtr,
                              TreeNode *& rightPtr) const
{
    leftPtr = nodePtr->leftChildPtr;
    rightPtr = nodePtr->rightChildPtr;
} // end getChildPtrs

void BinaryTree::setChildPtrs(TreeNode *nodePtr,
                              TreeNode *leftPtr,
                              TreeNode *rightPtr)
{
    nodePtr->leftChildPtr = leftPtr;
    nodePtr->rightChildPtr = rightPtr;
} // end setChildPtrs

```

```

void BinaryTree:: preOrder(TreeNode* r)
{
    if(r != NULL)
    {
        cout << r->item << " ";
        preOrder(r->leftChildPtr);
        preOrder(r->rightChildPtr);
    }//if
} //preOrder

void BinaryTree:: inOrder(TreeNode* r)
{
    if(r != NULL)
    {
        inOrder(r->leftChildPtr);
        cout << r->item << " ";
        inOrder(r->rightChildPtr);
    }//if
} //inOrder

void BinaryTree:: postOrder(TreeNode* r)
{
    if(r != NULL)
    {
        postOrder(r->leftChildPtr);
        postOrder(r->rightChildPtr);
        cout << r->item << " ";
    }//if
} //postOrder
// End of implementation routines
} // end of namespace csci301_BinaryTree

```