```
//////////////////////////////
//                          //
//                          //
//       BinaryTree.h       //
//                          //
//                          //
//////////////////////////////

#ifndef BINARY_TREE_H
#define BINARY_TREE_H

#include <iostream>   //Included for STL cout

struct TreeNode
{
    int data;         // The data in this node
    TreeNode *left;   // Pointer to the left subtree
    TreeNode *right;  // Pointer to the right subtree
}

class BinaryTree
{

      private:
            TreeNode *root;  //Root node of the binary tree
            void insertNode(TreeNode *nextNode, TreeNode *insertNode);/
*Insert node into the tree*/
            void removeNode(int data, TreeNode* parent);/*Removes node from
tree*/
            TreeNode* getLeftMost(TreeNode* node);/*Returns leftmost node of
a subtree*/
            void printInorder(TreeNode* node); /*Output the inorder of a
subtree*/
            void printPostorder(TreeNode *node); /*Output the postorder of a
subtree*/
            void printPreorder(); /*Output the preorder of a subtree*/

      public:
            BinaryTree();//Constructor
            ~BinaryTree();//Destructor
            void insertData(int dataToInsert);//Insert data into the tree
            void removeData(int dataToRemove);//Delete a node from the tree
            void printInorder(); //Output the inorder of the tree
            void printPostorder(); //Output the postorder of the tree
            void printPreorder(); //Output the preorder of the tree

};
#endif
//////////////////////////////
//                          //
//                          //
```

```
//      BinaryTree.cpp       //
//                           //
//                           //
///////////////////////////////

#include "BinaryTree.h"

/** Constructor of BinaryTree class **/
BinaryTree::BinaryTree()
{
      root = NULL;
}

/** Destructor of BinaryTree class **/
BinaryTree::~BinaryTree()
{
      delete root;
}

/** Inserts data into the BinaryTree
    @param the data to be inserted
 */
void BinaryTree::insertData(int dataToInsert)
{
      if(root == NULL)
            root->data = dataToInsert;
      else
      {
            TreeNode *newNode = NULL;
            newNode->data = dataToInsert;
            insertNode(root, newNode);
      }
}

/** Inserts a node into the binary tree
    @param the node we are inserting a node into
    @param the node being inserted into the tree
 */
void BinaryTree::insertNode(TreeNode *nextNode, TreeNode *insertNode)
{
      if(insertNode->data > nextNode->data)
      {
            if(nextNode->right == NULL)
                  nextNode->right = insertNode;
            else
                  insertNode(nextNode->right, insertNode);
      }

      if(insertNode->data <= nextNode->data)
      {
            if(nextNode->left == NULL)
```

```
                  nextNode->left = insertNode;
            else
                  insertNode(nextNode->left, insertNode);
      }
}

/** Removes data from the tree **/
void BinaryTree::removeData(int dataToRemove)
{
      removeNode(dataToRemove, root);
}

/** Removes a node from the tree
    @param the data contained in the node being removed
    @param the tree or subtree we are removing a node from
 */
void BinaryTree::removeNode(int data, TreeNode *parent)
{
      if(parent == NULL)
            return;
      if(parent->data == data)
      {
            if((parent->right == NULL || parent->right->right == NULL) &&
parent->left != NULL)
                  parent = left;
            if(parent->left = NULL && parent->right != NULL)
                  parent = right;
            if(parent->left == NULL && parent->right == NULL)
                  delete parent;
            parent = getLeftMost(parent->right);
            return;
      }
      if(parent->data <= data)
            removeNode(data, parent->left);
      if(parent->data > data)
            removeNode(data, parent->right);
}

/** Returns the leftmost node of a tree or subtree
    @param the tree or subtree to return the leftmost node from
    @return the leftmost node of a tree or subtree
 */
TreeNode* BinaryTree::getLeftMost(TreeNode* node)
{
      if(node->left == NULL)
            return node;
      else
            return getLeftMost(node->left);
}
```

```
/** Prints the inorder transversal of the binary tree **/
void BinaryTree::printInorder()
{
      printInorder(root);
}

/** Prints the inorder transversal of a tree or subtree
    @param the tree or subtree we are printing the inorder of
 */
void BinaryTree::printInorder(TreeNode *node)
{
    if (node != NULL)
    {
        printInorder(node->left);
        std::cout << node->item << " ";
        printInorder(node->right);
    }
}

/** Prints the postorder transversal of the binary tree **/
void BinaryTree::printPostorder()
{
      printPostorder(root);
}

/** Prints the postorder transversal of a tree or subtree
    @param the tree or subtree we are printing the postorder of
 */
void BinaryTree::printPostorder(TreeNode *node)
{
    if (node != NULL)
    {
        printPostorder(node->left);
        printPostorder(node->right);
        std::cout << node->item << " ";
    }
}

/** Prints the preorder transversal of the binary tree **/
void BinaryTree::printPreorder()
{
      printPreorder(root);
}

/** Prints the preorder transversal of a tree or subtree
    @param the tree or subtree we are printing the preorder of
 */
void BinaryTree::printPreorder(TreeNode *node)
{
    if (node != NULL)
    {
        std::cout << node->item << " ";
```

```
        printPreorder(node->left);
        printPreorder(node->right);
    }
}
```

```
////////////////////////////////
//                            //
//                            //
//          main.cpp          //
//                            //
//                            //
////////////////////////////////

#include <iostream> //Included for STL cout

#include "BinaryTree.h"

void Test1();
void Test2();
void Test3();

int main()
{
      Test1();
      Test2();
```

```
	Test3();

	return 0;
}

/** Insert nodes into a binary tree and print the transversals **/
void Test1()
{
	BinaryTree *binarytree = new BinaryTree();

	binaryTree->insertData(3);
	binaryTree->insertData(1);
	binaryTree->insertData(5);
	binaryTree->insertData(3);
	binaryTree->insertData(7);
	binaryTree->insertData(6);
	binaryTree->insertData(2);

	std::cout << "Inorder of tree : ";
	binaryTree->printInorder();
	std::cout << std::endl;
	std::cout << "Preorder of tree : ";
	binaryTree->printPreorder();
	std::cout << std::endl;
	std::cout << "Postorder of tree : ";
	binaryTree->printPostorder();
	std::cout << std::endl;
}

/** Insert nodes into a binary tree, delete a node, and print the
transversals **/
void Test2()
{
	BinaryTree *binarytree = new BinaryTree();

	binaryTree->insertData(5);
	binaryTree->insertData(6);
	binaryTree->insertData(4);
	binaryTree->insertData(2);
	binaryTree->insertData(7);
	binaryTree->insertData(9);
	binaryTree->insertData(12);

	binaryTree->deleteData(7);

	std::cout << "Inorder of tree : ";
	binaryTree->printInorder();
	std::cout << std::endl;
	std::cout << "Preorder of tree : ";
	binaryTree->printPreorder();
	std::cout << std::endl;
	std::cout << "Postorder of tree : ";
	binaryTree->printPostorder();
	std::cout << std::endl;
}
```

```
/** Insert nodes into a binary tree, delete the root node, and print the
transversals **/
void Test3()
{
      BinaryTree *binarytree = new BinaryTree();

      binaryTree->insertData(10);
      binaryTree->insertData(8);
      binaryTree->insertData(13);
      binaryTree->insertData(6);
      binaryTree->insertData(4);
      binaryTree->insertData(5);
      binaryTree->insertData(11);
      binaryTree->insertData(15);

      binaryTree->deleteData(10);

      std::cout << "Inorder of tree : ";
      binaryTree->printInorder();
      std::cout << std::endl;
      std::cout << "Preorder of tree : ";
      binaryTree->printPreorder();
      std::cout << std::endl;
      std::cout << "Postorder of tree : ";
      binaryTree->printPostorder();
      std::cout << std::endl;
}
```

Program Output:

```
Inorder of Tree : 1 2 3 3 5 6 7
Preorder of Tree : 3 1 2 3 5 6 7
Postorder of Tree : 2 3 1 6 5 7 3
Inorder of Tree : 2 4 5 6 9 12
Preorder of Tree : 5 4 2 6 9 12
Postorder of Tree : 2 4 12 9 6 5
Inorder of Tree : 4 5 6 8 11 13 15
Preorder of Tree : 11 8 6 4 5 13 15
Postorder of Tree : 5 4 6 8 15 13 11
```