#include <iostream>

using namespace std;

struct node

{

int data;

node\* right;

node\* left;

};

class BST // Binary Search Tree

{

private:

void insert(node\*, int);

bool search(int, node\*);// if found or not

void inorder(node\*); // minimum to maximum

void preorder(node\*);

void postorder(node\*);

void deleteByData(node\*, int);

public:

node\* root;

BST()

{

root = NULL;

}

int min();

int max();

void insert(int);

bool search(int);

void inorder();

void preorder();

void postorder();

void deleteNode(node\*);

void deleteByData(int);

};

int BST::min()

{

if (root != nullptr)

{

node\* temp = root;

int min = 0;

while (temp->left != nullptr)

{

temp = temp->left;

}

min = temp->data;

return min;

}

else

{

cout << "\n There is not an element in the binary tree." << endl;

return 0;

}

}

int BST::max()

{

if (root != nullptr)

{

node\* temp = root;

int max = 0;

while (temp->right != nullptr)

{

temp = temp->right;

}

max = temp->data;

return max;

}

else

{

cout << "\n There is not an element in the binary tree." << endl;

return 0;

}

}

void BST::insert(node\* n, int data)

{

if (data < n->data)

{

if (n->left == nullptr)

{

n->left = new node;

n->left->data = data;

n->left->right = nullptr;

n->left->left = nullptr;

}

else

{

insert(n->left, data);

}

}

else if (data >= n->data)

{

if (n->right == NULL)

{

n->right = new node;

n->right->data = data;

n->right->right = nullptr;

n->right->left = nullptr;

}

else

{

insert(n->right, data);

}

}

}

void BST::insert(int data)

{

if (root != nullptr)

{

insert(root, data);

}

else

{

root = new node;

root->data = data;

root->right = nullptr;

root->left = nullptr;

}

}

void BST::inorder(node\* n)

{

if (n != nullptr)

{

inorder(n->left);

cout << n->data << endl;

inorder(n->right);

}

}

void BST::inorder()

{

if (root != nullptr)

{

inorder(root);

}

else

{

cout << "\n There is not an element in the binary tree." << endl;

}

}

void BST::preorder(node\* n)

{

if (n != nullptr)

{

cout << n->data << endl;

preorder(n->left);

preorder(n->right);

}

}

void BST::preorder()

{

if (root == NULL)

{

cout << "\n There is not an element in the binary tree." << endl;

}

else

{

preorder(root);

}

}

void BST::postorder(node\* n)

{

if (n != nullptr)

{

cout << n->data << endl;

postorder(n->right);

postorder(n->left);

}

}

void BST::postorder()

{

if (root == NULL)

{

cout << "\n There is not an element in the binary tree." << endl;

}

else

{

postorder(root);

}

}

bool BST::search(int data, node\* n)

{

bool hasFound = false;

if (n == nullptr)

{

return false;

}

if (data == n->data)

{

return true;

}

else if (data < n->data)

{

hasFound = search(data, n->left);

}

else

{

hasFound = search(data, n->right);

}

return hasFound;

}

bool BST::search(int data)

{

if (root == nullptr)

{

cout << "\n There is not an element in the binary tree." << endl;

}

else

{

return search(data, root);

}

}

//void BST::deleteNode(node\* node)

//{

// delete node;

// node->left = nullptr;

// node->right= nullptr;

//}

//

//void BST::deleteByData(node\* root, int data)

//{

// if (root->data == data)

// {

// node\* n = root->right;

// n->right = root->right;

// n.led

// root = root->right;

// deleteNode(root);

// root->left = root->left;

//

//

// }

//

//

//}

//

//void BST::deleteByData(int data)

//{

// if (root != nullptr)

// {

// if (search(data))

// {

// deleteByData(root,data);

// }

// else

// {

// cout << data << " does not exist in the binary tree!" << endl;

// }

// }

// else

// {

// cout << "\n There is not an element in the binary tree." << endl;

// }

//}

int main()

{

BST tree;

tree.insert(10);

tree.insert(8);

tree.insert(9);

tree.insert(23);

tree.insert(16);

tree.insert(34);

tree.insert(5);

tree.inorder();

//tree.preorder();

//tree.postorder();

//int minimum = tree.min();

//int maximum = tree.max();

//cout << "Minimum value of the tree : " << minimum << endl;

//cout << "Maximum value of the tree : " << maximum << endl;

return 0;

}