# Mahad Ghauri

233523

DSA Lab Task

#include <iostream>

#include <string>

using namespace std;

class Node

{

private:

int data;

Node \*left, \*right;

public:

Node()

{

}

Node(int x)

{

data = x;

left = right = NULL;

}

void setData(int value)

{

data = value;

}

int getData()

{

return data;

}

void setLeft(Node \*other)

{

left = other;

}

Node \*getLeft()

{

return left;

}

void setRight(Node \*other)

{

right = other;

}

Node \*getRight()

{

return right;

}

};

Node \*insertNode(Node \*root, int target)

{

if (!root)

{

Node \*temp = new Node(target);

return temp;

}

// The desired value is smaller than the root node

if (target < root->getData())

{

root->setLeft(insertNode(root->getLeft(), target));

}

else

{

root->setRight(insertNode(root->getRight(), target));

}

return root;

}

bool searchNode(Node \*root, int target)

{

if (!root)

{

return 0;

}

if (root->getData() == target)

{

return 1;

}

if (root->getData() > target)

{

return searchNode(root->getLeft(), target);

}

else

{

return searchNode(root->getRight(), target);

}

}

Node \*deleteNode(Node \*root, int target)

{

if (!root)

{

return NULL;

}

if (root->getData() > target)

{

root->setLeft(deleteNode(root->getLeft(), target));

return root;

}

else if (root->getData() < target)

{

root->setRight(deleteNode(root->getRight(), target));

return root;

}

else

{

if (!root->getLeft() && !root->getRight()) // Leaf Node

{

delete root;

return NULL;

}

else if (!root->getRight()) // Left child exists

{

Node \*temp = root->getLeft();

delete root;

return temp;

}

else if (!root->getLeft()) // Right child exists

{

Node \*temp = root->getRight();

delete root;

return temp;

}

else // Both child exists

{

Node \*child = root->getLeft();

Node \*parent = root;

while (root->getRight()) // to get the highest number in the left side of the root node

{

parent = child;

child = child->getRight();

}

if (root != parent)

{

parent->setRight(child->getLeft());

child->setLeft(root->getLeft());

child->setRight(root->getRight());

delete root;

return child;

}

else // if the rooot node is equal to the parent node then this cae happend

{

child->setRight(root->getRight());

delete root;

return child;

}

}

}

}

int minimumValue(Node \*root)

{

if (root == NULL)

return -1;

Node \*current = root;

while (current->getLeft())

{

current = current->getLeft();

}

return current->getData();

}

int maximumValue(Node \*root)

{

if (root == NULL)

{

return -1;

}

Node \*current = root;

while (current->getRight())

{

current = current->getRight();

}

return current->getData();

}

void inOrder(Node \*root)

{

if (root == NULL)

{

return;

}

inOrder(root->getLeft());

cout << root->getData() << " ";

inOrder(root->getRight());

}

void preOrder(Node \*root)

{

if (root == NULL)

return;

cout << " " << root->getData();

preOrder(root->getLeft());

preOrder(root->getRight());

}

void postOrder(Node \*root)

{

if (root == NULL)

{

return;

}

postOrder(root->getLeft());

postOrder(root->getRight());

cout << root->getData() << " ";

}

int main()

{

Node \*root = NULL;

int choice, value;

cout << "Binary Search Tree Program\n";

while (true)

{

cout << "\n -------- Menu --------:\n";

cout << " 1 --> Insert\n";

cout << " 2 --> Search\n";

cout << " 3 --> Delete\n";

cout << " 4 --> Find Minimum\n";

cout << " 5 --> Find Maximum\n";

cout << " 6 --> Inorder Traversal\n";

cout << " 7 --> Preorder Traversal\n";

cout << " 8 --> Postorder Traversal\n";

cout << " 9 --> Exit\n";

cout << "\n -------- Enter your choice --------: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to insert: ";

cin >> value;

root = insertNode(root, value);

cout << "Inserted " << value << " into the tree.\n";

break;

case 2:

cout << "Enter value to search: ";

cin >> value;

if (searchNode(root, value))

{

cout << "Value " << value << " found in the tree.\n";

}

else

{

cout << "Value " << value << " not found in the tree.\n";

}

break;

case 3:

cout << "Enter value to delete: ";

cin >> value;

root = deleteNode(root, value);

cout << "Deleted " << value << " from the tree (if it existed).\n";

break;

case 4:

value = minimumValue(root);

if (value != -1)

{

cout << "Minimum value in the tree: " << value << endl;

}

else

{

cout << "Tree is empty!\n";

}

break;

case 5:

value = maximumValue(root);

if (value != -1)

{

cout << "Maximum value in the tree: " << value << endl;

}

else

{

cout << "Tree is empty!\n";

}

break;

case 6:

cout << "Inorder traversal: ";

inOrder(root);

cout << endl;

break;

case 7:

cout << "Preorder traversal: ";

preOrder(root);

cout << endl;

break;

case 8:

cout << "Postorder traversal: ";

postOrder(root);

cout << endl;

break;

case 9:

cout << "Exiting...\n";

return 0;

default:

cout << "Invalid choice, please try again.\n";

break;

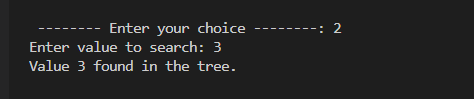
}

}

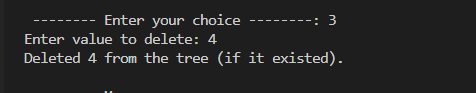
return 0;

}

**Task 1:**



**Task 2:**



**Task 3** **:**

