#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*left;

Node \*right;

bool leftThread;

bool rightThread;

Node(int value)

{

data = value;

left = right = nullptr;

leftThread = rightThread = true;

}

};

class ThreadedBinaryTree

{

private:

Node \*root;

Node \*inorderSuccessor(Node \*ptr)

{

if (ptr->rightThread)

{

return ptr->right;

}

ptr = ptr->right;

while (!ptr->leftThread)

{

ptr = ptr->left;

}

return ptr;

}

Node \*leftmost(Node \*node)

{

if (node == nullptr)

{

return nullptr;

}

while (node->left != nullptr && !node->leftThread)

{

node = node->left;

}

return node;

}

public:

ThreadedBinaryTree() : root(nullptr) {}

void insert(int value)

{

Node \*newNode = new Node(value);

if (root == nullptr)

{

root = newNode;

return;

}

Node \*current = root;

Node \*parent = nullptr;

while (current != nullptr)

{

parent = current;

if (value < current->data)

{

if (!current->leftThread)

{

current = current->left;

}

else

{

break;

}

}

else

{

if (!current->rightThread)

{

current = current->right;

}

else

{

break;

}

}

}

if (value < parent->data)

{

newNode->left = parent->left;

newNode->right = parent;

parent->leftThread = false;

parent->left = newNode;

}

else

{

newNode->left = parent;

newNode->right = parent->right;

parent->rightThread = false;

parent->right = newNode;

}

}

void inorderTraversal()

{

if (root == nullptr)

{

cout << "Tree is empty" << endl;

return;

}

cout << "Inorder Traversal: ";

Node \*current = leftmost(root);

while (current != nullptr)

{

cout << current->data << " ";

if (current->rightThread)

{

current = current->right;

}

else

{

current = leftmost(current->right);

}

}

cout << endl;

}

void preorderTraversal()

{

if (root == nullptr)

{

cout << "Tree is empty" << endl;

return;

}

cout << "Preorder Traversal: ";

Node \*current = root;

while (current != nullptr)

{

cout << current->data << " ";

if (!current->leftThread)

{

current = current->left;

}

else if (!current->rightThread)

{

current = current->right;

}

else

{

while (current != nullptr && current->rightThread)

{

current = current->right;

}

if (current != nullptr)

{

current = current->right;

}

}

}

cout << endl;

}

void destroyTree(Node \*node)

{

if (node == nullptr)

return;

if (!node->leftThread)

{

destroyTree(node->left);

}

if (!node->rightThread)

{

destroyTree(node->right);

}

delete node;

}

~ThreadedBinaryTree()

{

destroyTree(root);

}

};

int main()

{

ThreadedBinaryTree tree;

int choice, value;

do

{

cout << "\nThreaded Binary Tree Operations:\n";

cout << "1. Insert Node\n";

cout << "2. Inorder Traversal\n";

cout << "3. Preorder Traversal\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to insert: ";

cin >> value;

tree.insert(value);

cout << "Node inserted successfully!\n";

break;

case 2:

tree.inorderTraversal();

break;

case 3:

tree.preorderTraversal();

break;

case 4:

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

break;

default:

cout << "Invalid choice! Please try again.\n";

}

} while (choice != 4);

return 0;

}