#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct Node\* insert(struct Node\* root, int data) {

if (root == NULL) {

root = createNode(data);

} else if (data < root->data) {

root->left = insert(root->left, data);

} else if (data > root->data) {

root->right = insert(root->right, data);

}

return root;

}

struct Node\* findMin(struct Node\* root) {

while (root && root->left != NULL) {

root = root->left;

}

return root;

}

struct Node\* deleteNode(struct Node\* root, int data) {

if (root == NULL) {

printf("The value to be deleted is not present inside the tree\n");

return root;

}

if (data < root->data) {

root->left = deleteNode(root->left, data);

} else if (data > root->data) {

root->right = deleteNode(root->right, data);

} else {

// Node with one child or no child

if (root->left == NULL) {

struct Node\* temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

struct Node\* temp = root->left;

free(root);

return temp;

}

// Node with two children

struct Node\* temp = findMin(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

struct Node\* search(struct Node\* root, int data) {

if (root == NULL || root->data == data)

return root;

if (data < root->data)

return search(root->left, data);

else

return search(root->right, data);

}

// Preorder traversal

void preorder(struct Node\* root) {

if (root != NULL) {

printf("%d ", root->data);

preorder(root->left);

preorder(root->right);

}

}

// Inorder traversal

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

printf("%d ", root->data);

inorder(root->right);

}

}

// Postorder traversal

void postorder(struct Node\* root) {

if (root != NULL) {

postorder(root->left);

postorder(root->right);

printf("%d ", root->data);

}

}

int main() {

struct Node\* root = NULL;

int choice, value;

struct Node\* foundNode;

while (1) {

printf("\nBinary Search Tree Operations:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Search\n");

printf("4. Preorder Traversal\n");

printf("5. Inorder Traversal\n");

printf("6. Postorder Traversal\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

root = insert(root, value);

break;

case 2:

if (root == NULL)

printf("Tree is empty.\n");

else{

printf("Enter value to delete: ");

scanf("%d", &value);

root = deleteNode(root, value);

}

break;

case 3:

if (root == NULL)

{

printf("Tree is empty.\n");

}

else

{

printf("Enter value to search: ");

scanf("%d", &value);

foundNode = search(root, value);

if (foundNode != NULL)

{

printf("Value %d found in the tree.\n", value);

}

else

{

printf("Value %d not found in the tree.\n", value);

}

}

break;

case 4:

if (root == NULL)

{

printf("Tree is empty.\n");

}

else {

printf("Preorder traversal: ");

preorder(root);

printf("\n");

}

break;

case 5:

if (root == NULL)

{

printf("Tree is empty.\n");

}

else {

printf("Inorder traversal: ");

inorder(root);

printf("\n");

}

break;

case 6:

if (root == NULL)

{

printf("Tree is empty.\n");

}

else {

printf("Postorder traversal: ");

postorder(root);

printf("\n");

}

break;

case 7:

exit(0);

default:

printf("Invalid choice! Please try again.\n");

}

}

return 0;

}