Binary Search Tree Implementation in C

Problem Statement: Implement a binary search tree (BST) in C and perform operations such as insertion, deletion, searching, and tree traversals (in-order, pre-order, post-order).

Objective: - Understand the structure and properties of a BST. - Implement basic operations on a BST using pointers. - Learn tree traversal techniques.

Algorithm:

1. **Insert Operation:**
    - If the tree is empty, create a new node and make it the root.
    - If the value is less than the current node's key, go to the left subtree.
    - If the value is greater than the current node's key, go to the right subtree.
    - Repeat recursively until the proper position is found.
2. **Search Operation:**
    - If the tree is empty or the current node key equals the target, return the node.
    - If target is greater than current node key, search in the right subtree.
    - If target is smaller, search in the left subtree.
3. **Delete Operation:**
    - Locate the node to delete.
    - If node has no child, remove it directly.
    - If node has one child, replace it with its child.
    - If node has two children, find the minimum value in the right subtree, replace the node's key with it, and delete the minimum node recursively.
4. **Traversal Operations:**
    - **In-order:** Traverse left subtree, visit node, traverse right subtree.
    - **Pre-order:** Visit node, traverse left subtree, traverse right subtree.
    - **Post-order:** Traverse left subtree, traverse right subtree, visit node.

Program Code:

```
#include <stdio.h>
#include <stdlib.h>
struct BinaryTreeNode {
    int key;
    struct BinaryTreeNode *left, *right;
};
struct BinaryTreeNode* newNodeCreate(int value) {
    struct BinaryTreeNode* temp = (struct
BinaryTreeNode*)malloc(sizeof(struct BinaryTreeNode));
    temp->key = value;
    temp->left = temp->right = NULL;
    return temp;
}
```

```c
struct BinaryTreeNode* searchNode(struct BinaryTreeNode* root, int target) {
    if (root == NULL || root->key == target) return root;
    if (root->key < target) return searchNode(root->right, target);
    return searchNode(root->left, target);
}

struct BinaryTreeNode* insertNode(struct BinaryTreeNode* node, int value) {
    if (node == NULL) return newNodeCreate(value);
    if (value < node->key) node->left = insertNode(node->left, value);
    else if (value > node->key) node->right = insertNode(node->right, value);
    return node;
}

void postOrder(struct BinaryTreeNode* root) {
    if (root != NULL) {
        postOrder(root->left);
        postOrder(root->right);
        printf(" %d ", root->key);
    }
}

void inOrder(struct BinaryTreeNode* root) {
    if (root != NULL) {
        inOrder(root->left);
        printf(" %d ", root->key);
        inOrder(root->right);
    }
}

void preOrder(struct BinaryTreeNode* root) {
    if (root != NULL) {
        printf(" %d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}
struct BinaryTreeNode* findMin(struct BinaryTreeNode* root) {
    if (root == NULL) return NULL;
    else if (root->left != NULL) return findMin(root->left);
    return root;
}
struct BinaryTreeNode* delete(struct BinaryTreeNode* root, int x) {
    if (root == NULL) return NULL;
    if (x > root->key) root->right = delete(root->right, x);
    else if (x < root->key) root->left = delete(root->left, x);
    else {
        if (root->left == NULL && root->right == NULL) { free(root); return
NULL; }
```

```
        else if (root->left == NULL || root->right == NULL) {
            struct BinaryTreeNode* temp = (root->left == NULL) ? root->right
: root->left;
            free(root);
            return temp;
        } else {
            struct BinaryTreeNode* temp = findMin(root->right);
            root->key = temp->key;
            root->right = delete(root->right, temp->key);
        }
    }
    return root;
}
int main() {
    struct BinaryTreeNode* root = NULL;
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);

    if (searchNode(root, 60) != NULL) printf("60 found\n");
    else printf("60 not found\n");

    postOrder(root);
    printf("\n");
    preOrder(root);
    printf("\n");
    inOrder(root);
    printf("\n");

    root = delete(root, 70);
    printf("After Delete: \n");
    inOrder(root);
    printf("\n");

    return 0;
}
```

Sample Output:

```
60 found
20 40 30 60 80 70 50
50 30 20 40 70 60 80
20 30 40 50 60 70 80
After Delete:
20 30 40 50 60 80
```