```c
#include <stdio.h>
#include <stdlib.h>

// Define a structure for a binary tree node
struct BinaryTreeNode {
    int key;
    struct BinaryTreeNode *left, *right;
};

// Function to create a new node with a given value
struct BinaryTreeNode* newNodeCreate(int value)
{
    struct BinaryTreeNode* temp
        = (struct BinaryTreeNode*)malloc(
            sizeof(struct BinaryTreeNode));
    temp->key = value;
    temp->left = temp->right = NULL;
    return temp;
}

// Function to search for a node with a specific key in the
// tree
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);
}

// Function to insert a node with a specific value in the
// tree
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;
}

// Function to perform post-order traversal
void postOrder(struct BinaryTreeNode* root)
{
    if (root != NULL) {
        postOrder(root->left);
        postOrder(root->right);
        printf(" %d ", root->key);
    }
}

// Function to perform in-order traversal
void inOrder(struct BinaryTreeNode* root)
{
    if (root != NULL) {
        inOrder(root->left);
        printf(" %d ", root->key);
        inOrder(root->right);
    }
}

// Function to perform pre-order traversal
void preOrder(struct BinaryTreeNode* root)
{
    if (root != NULL) {
        printf(" %d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}

// Function to find the minimum value
struct BinaryTreeNode* findMin(struct BinaryTreeNode* root)
{
    if (root == NULL) {
        return NULL;
    }
    else if (root->left != NULL) {
        return findMin(root->left);
    }
    return root;
}

// Function to delete a node from the tree
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;
            if (root->left == NULL) {
                temp = root->right;
            }
            else {
                temp = 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()
{
    // Initialize the root node
    struct BinaryTreeNode* root = NULL;
    int key;

    // Insert nodes into the binary search tree
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);

    // Search for a node with key 60
    printf("\nEnter the number which you want to search : ");
    scanf("%d",&key);

    if (searchNode(root, key) != NULL) {
        printf("%d is found in the Tree.",key);
    }
    else {
        printf("%d is not found in the Tree.",key);
    }

    printf("\n");

    // Perform post-order traversal
    printf("\nThe post-order traversal of the given binary tree is - \n");
    postOrder(root);
    printf("\n");

    // Perform pre-order traversal
    printf("\nThe pre-order traversal of the given binary tree is - \n");
    preOrder(root);
    printf("\n");

    // Perform in-order traversal
    printf("\nThe in-order traversal of the given binary tree is - \n");
    inOrder(root);
    printf("\n");

    // Perform delete the node (70)
    struct BinaryTreeNode* temp = delete (root, 70);
    printf("\nAfter Delete: \n");
    inOrder(root);

    // Free allocated memory (not done in this code, but
    // good practice in real applications)

    return 0;
}
```