Input code :

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

// Define the structure of a tree node
struct Node {
    int key;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* createNode(int key) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->key = key;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Insertion function for BST
struct Node* insert(struct Node* root, int key) {
    if (root == NULL) {
        return createNode(key);
    }

    if (key < root->key) {
        root->left = insert(root->left, key);
    } else if (key > root->key) {
        root->right = insert(root->right, key);
    }

    return root;
}

// Function to find the minimum value node in a BST
struct Node* minValueNode(struct Node* root) {
    struct Node* current = root;
    while (current && current->left != NULL) {
        current = current->left;
    }
    return current;
}

// Deletion function for BST
struct Node* delete(struct Node* root, int key) {
    if (root == NULL) {
        return root;
    }

    // Recursively find the node to delete
    if (key < root->key) {
```

```
        root->left = delete(root->left, key);
    } else if (key > root->key) {
        root->right = delete(root->right, key);
    } else {
        // Node with only 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: Get the inorder successor (smallest in the right subtree)
        struct Node* temp = minValueNode(root->right);

        // Copy the inorder successor's content to this node
        root->key = temp->key;

        // Delete the inorder successor
        root->right = delete(root->right, temp->key);
    }

    return root;
}

// Search function for BST
struct Node* search(struct Node* root, int key) {
    if (root == NULL || root->key == key) {
        return root;
    }

    if (key < root->key) {
        return search(root->left, key);
    }

    return search(root->right, key);
}

// Function to print inorder traversal of the BST
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}

int main() {
    struct Node* root = NULL;
```

```
    // Insert nodes
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

    // Inorder traversal of the BST
    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");

    // Search operation
    int key = 40;
    if (search(root, key) != NULL) {
        printf("Node %d found in the BST\n", key);
    } else {
        printf("Node %d not found in the BST\n", key);
    }

    // Deletion operation
    root = delete(root, 20);
    printf("Inorder traversal after deletion: ");
    inorder(root);
    printf("\n");

    return 0;
}
```

Output :

Inorder traversal: 20 30 40 50 60 70 80
Node 40 found in the BST
Inorder traversal after deletion: 30 40 50 60 70 80