Experiment No. 8

```
/* Implement various operations on a Binary Search Tree, such as insertion, deletion, display,
and search.*/
#include <stdio.h>
#include <stdlib.h>

// Structure for a node in BST
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

// Function to insert a node into BST
struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);

    return root;
}

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

// Function to delete a node from BST
struct Node* deleteNode(struct Node* root, int value) {
    if (root == NULL)
        return root;

    if (value < root->data)
        root->left = deleteNode(root->left, value);
    else if (value > root->data)
        root->right = deleteNode(root->right, value);
    else {
        // Node found
        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;
}

// Function to search for a node in BST
struct Node* search(struct Node* root, int key) {
    if (root == NULL || root->data == key)
        return root;

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

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

int main() {
    struct Node* root = NULL;
    int choice, value;
    struct Node* result;

    while (1) {
        printf("\n\n====== Binary Search Tree Operations ======\n");
        printf("1. Insert Node\n");
        printf("2. Delete Node\n");
        printf("3. Search Node\n");
        printf("4. Display (Inorder Traversal)\n");
        printf("5. 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);
        printf("Node inserted successfully!\n");
        break;

    case 2:
        printf("Enter value to delete: ");
        scanf("%d", &value);
        root = deleteNode(root, value);
        printf("Node deleted successfully (if it existed)!\n");
        break;

    case 3:
        printf("Enter value to search: ");
        scanf("%d", &value);
        result = search(root, value);
        if (result != NULL)
            printf("Node %d found in the tree.\n", value);
        else
            printf("Node %d not found!\n", value);
        break;
```

```
            case 4:
                printf("Inorder Traversal of BST: ");
                inorder(root);
                printf("\n");
                break;

            case 5:
                printf("Exiting...\n");
                exit(0);

            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

Output:-

```
====== Binary Search Tree Operations ======
1. Insert Node
2. Delete Node
3. Search Node
4. Display (Inorder Traversal)
5. Exit
```

```
Enter your choice: 1

Enter value to insert: 63

Node inserted successfully!


====== Binary Search Tree Operations ======

1. Insert Node

2. Delete Node

3. Search Node

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 1

Enter value to insert: 79

Node inserted successfully!


====== Binary Search Tree Operations ======

1. Insert Node

2. Delete Node

3. Search Node

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 1

Enter value to insert: 79

Node inserted successfully!


====== Binary Search Tree Operations ======

1. Insert Node
```

```
2. Delete Node
3. Search Node
4. Display (Inorder Traversal)
5. Exit
Enter your choice: 1
Enter value to insert: 56
Node inserted successfully!

====== Binary Search Tree Operations ======
1. Insert Node
2. Delete Node
3. Search Node
4. Display (Inorder Traversal)
5. Exit
Enter your choice: 3
Enter value to search: 56
Node 56 found in the tree.

====== Binary Search Tree Operations ======
1. Insert Node
2. Delete Node
3. Search Node
4. Display (Inorder Traversal)
5. Exit
Enter your choice: 2
Enter value to delete: 79
```

```
Node deleted successfully (if it existed)!


====== Binary Search Tree Operations ======

1. Insert Node

2. Delete Node

3. Search Node

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 4

Inorder Traversal of BST: 56 63


====== Binary Search Tree Operations ======

1. Insert Node

2. Delete Node

3. Search Node

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 5

Exiting...


=== Code Execution Successful ===
```