

EX NO :9	BINARY SEARCH TREE
19.9.2025	
2403717610421101	

AIM: Construct a BST and perform the following operations.

PROGRAM:

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

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

```
struct Node {
    int data;
    struct Node *left, *right;
};
```

```
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
```

```
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;
}
```

```

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

```

```

struct Node* findMin(struct Node* root) {
    while (root && root->left != NULL)
        root = root->left;
    return root;
}

```

```

struct Node* findMax(struct Node* root) {
    while (root && root->right != NULL)
        root = root->right;
    return root;
}

```

```

struct Node* deleteNode(struct Node* root, int key) {
    if (root == NULL) {
        printf("Error: Tree is empty, cannot delete!\n");
        return NULL;
    }
    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {

```

```

    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;
    }
    struct Node* temp = findMin(root->right);
    root->data = temp->data;
    root->right = deleteNode(root->right, temp->data);
}
return root;
}

```

```

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;

    while (1) {
        printf("\n\n--- Binary Search Tree Operations ---\n");
        printf("1. Insert\n2. Delete\n3. Search\n4. FindMin\n5. FindMax\n6. Display (Inorder)\n7. 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("Error: Tree is empty, nothing to delete!\n");
        } else {
            printf("Enter value to delete: ");
            scanf("%d", &value);
            root = deleteNode(root, value);
        }
        break;
    case 3:
        printf("Enter value to search: ");
        scanf("%d", &value);
        if (search(root, value))
            printf("Element %d found!\n", value);
        else
            printf("Element %d not found!\n", value);
        break;
    case 4:
        if (root)
            printf("Minimum value: %d\n", findMin(root)->data);
        else
            printf("Tree is empty!\n");
        break;
```

```
case 5:
    if (root)
        printf("Maximum value: %d\n", findMax(root)->data);
    else
        printf("Tree is empty!\n");
    break;
case 6:
    if (root) {
        printf("Inorder Traversal: ");
        inorder(root);
        printf("\n");
    } else {
        printf("Tree is empty!\n");
    }
    break;
case 7:
    printf("Task completed");
    exit(0);
default:
    printf("Invalid choice!\n");
}
}
return 0;
}
```

OUTPUT:

```
--- Binary Search Tree Operations ---
1. Insert
2. Delete
3. Search
4. FindMin
5. FindMax
6. Display (Inorder)
7. Exit
Enter your choice: 6
Inorder Traversal: 45 67
```

```
--- Binary Search Tree Operations ---
1. Insert
2. Delete
3. Search
4. FindMin
5. FindMax
6. Display (Inorder)
7. Exit
Enter your choice: 7
Task completed
```

RESULT:

The program was successfully implemented, compiled, and executed. It performs the desired operations as per the problem statement using appropriate data structures and algorithms, and produces the correct output for the given input.