CHRIST

(DEEMED TO BE UNIVERSITY)

DELHI-NCR, INDIA

Data Structure and Algorithm (MCA 271)

Lab Practical –

BY

Himanshu Heda (24225013)

SUBMITTED TO

Prof. Vandna Kansal

SCHOOL OF SCIENCES

2024-2025

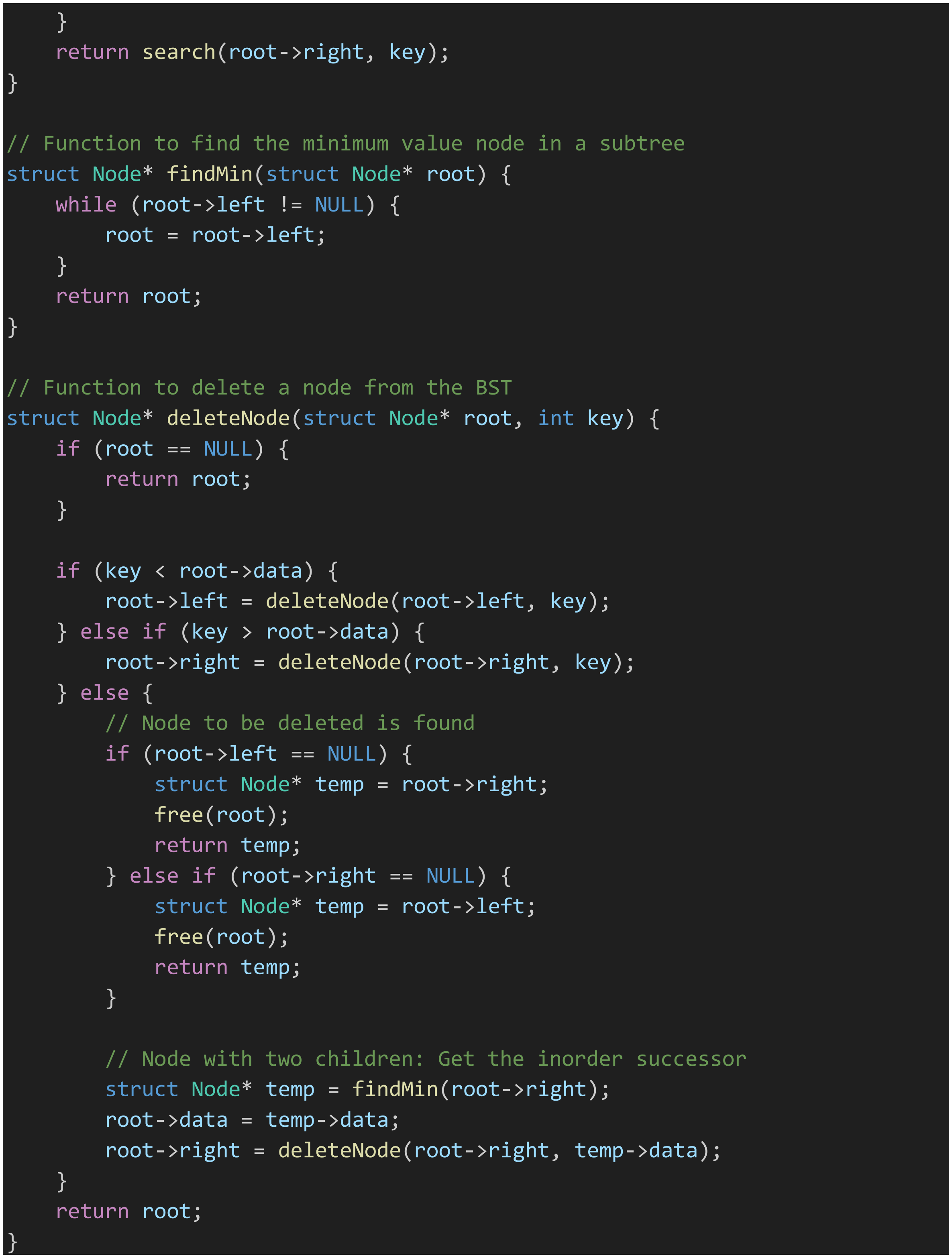
Program Description:

Code of the program

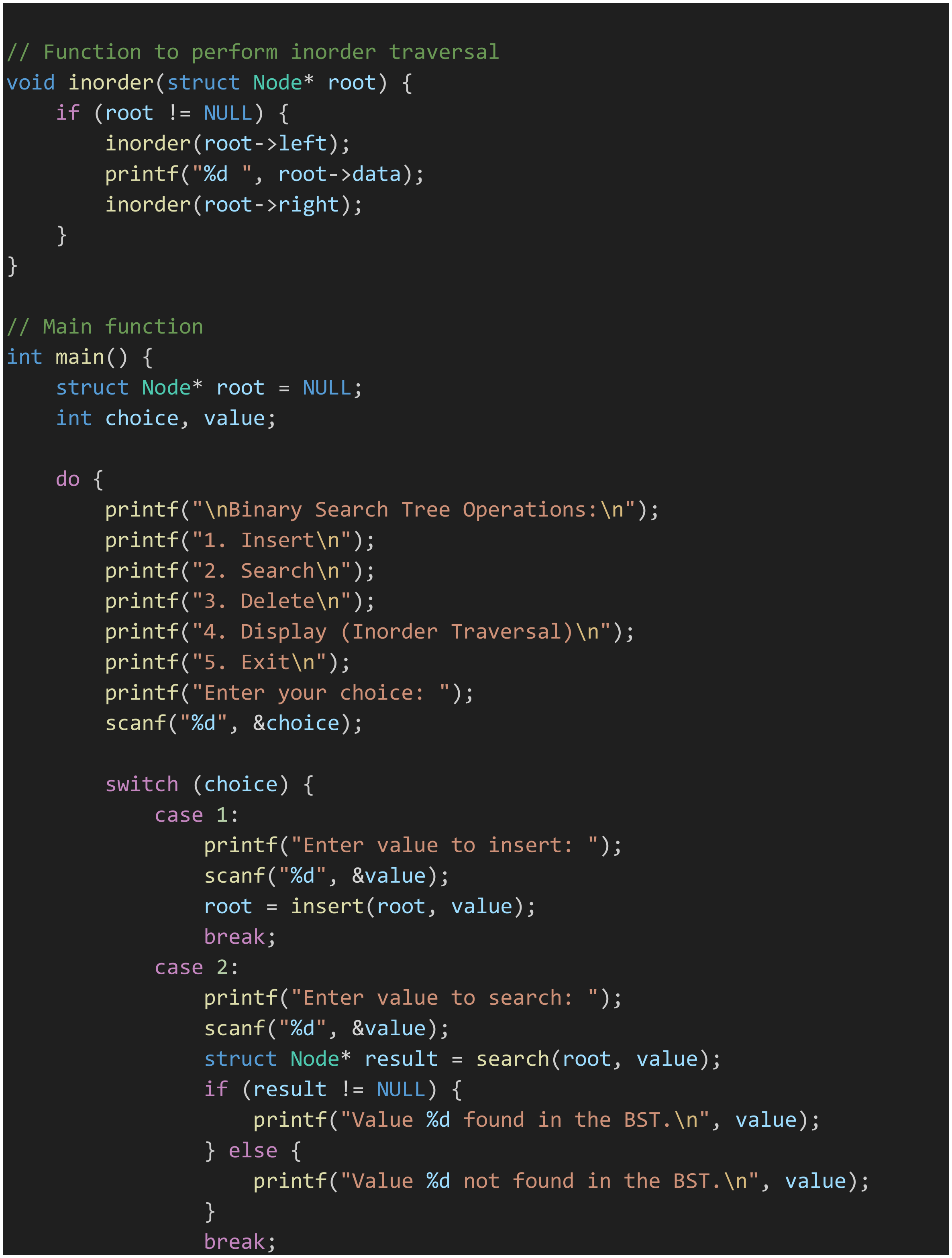
Output: - Paste the o/p of the program.

1. Tree Traversal : --

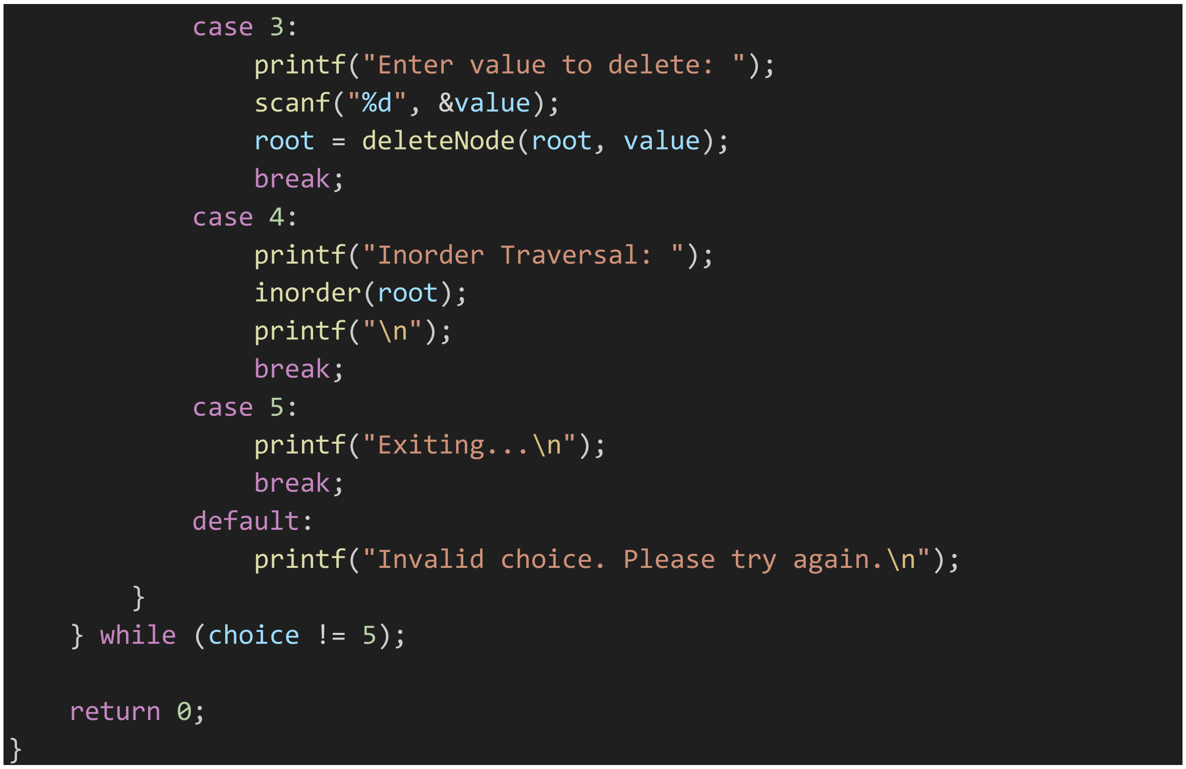
```
#include <stdio.h>
#include <stdlib.h>

// Define the structure for a node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

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

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

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

    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    } else {
        // Node to be deleted is 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: Get the inorder successor
        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
```

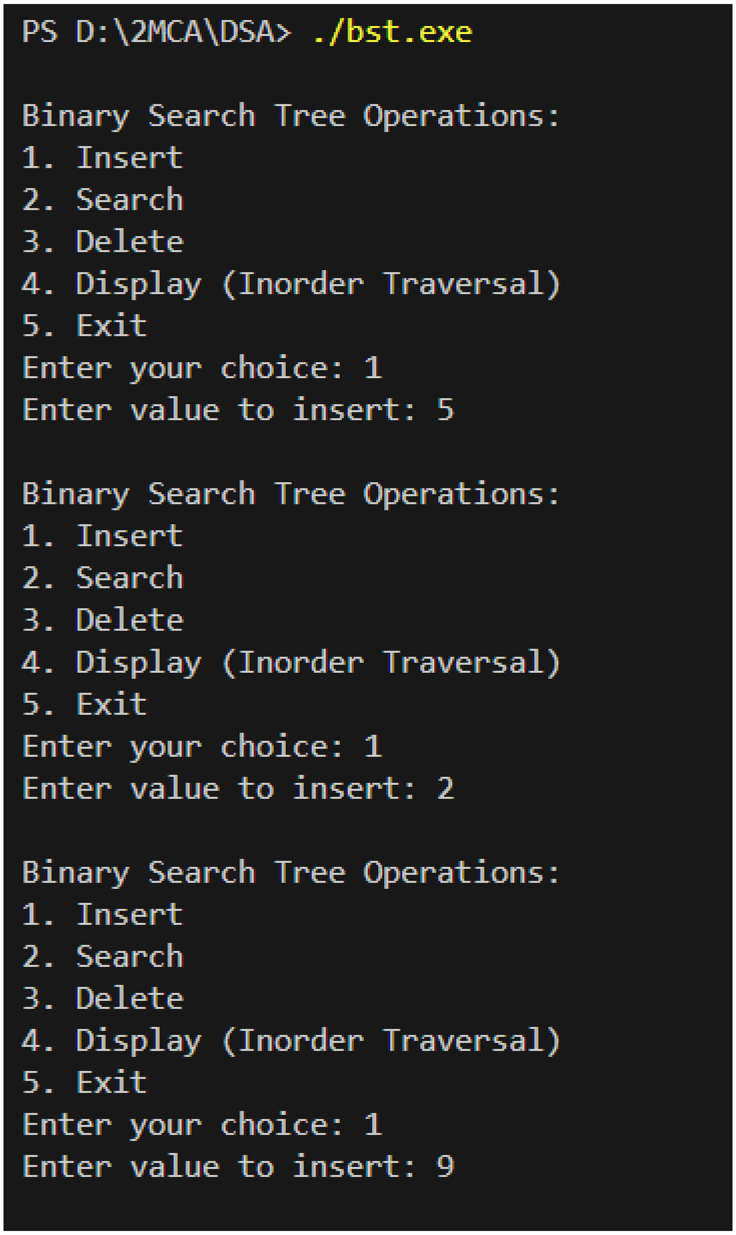


```c
// Function to perform inorder traversal
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

// Main function
int main() {
    struct Node* root = NULL;
    int choice, value;

    do {
        printf("\nBinary Search Tree Operations:\n");
        printf("1. Insert\n");
        printf("2. Search\n");
        printf("3. Delete\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);
                break;
            case 2:
                printf("Enter value to search: ");
                scanf("%d", &value);
                struct Node* result = search(root, value);
                if (result != NULL) {
                    printf("Value %d found in the BST.\n", value);
                } else {
                    printf("Value %d not found in the BST.\n", value);
                }
                break;
```

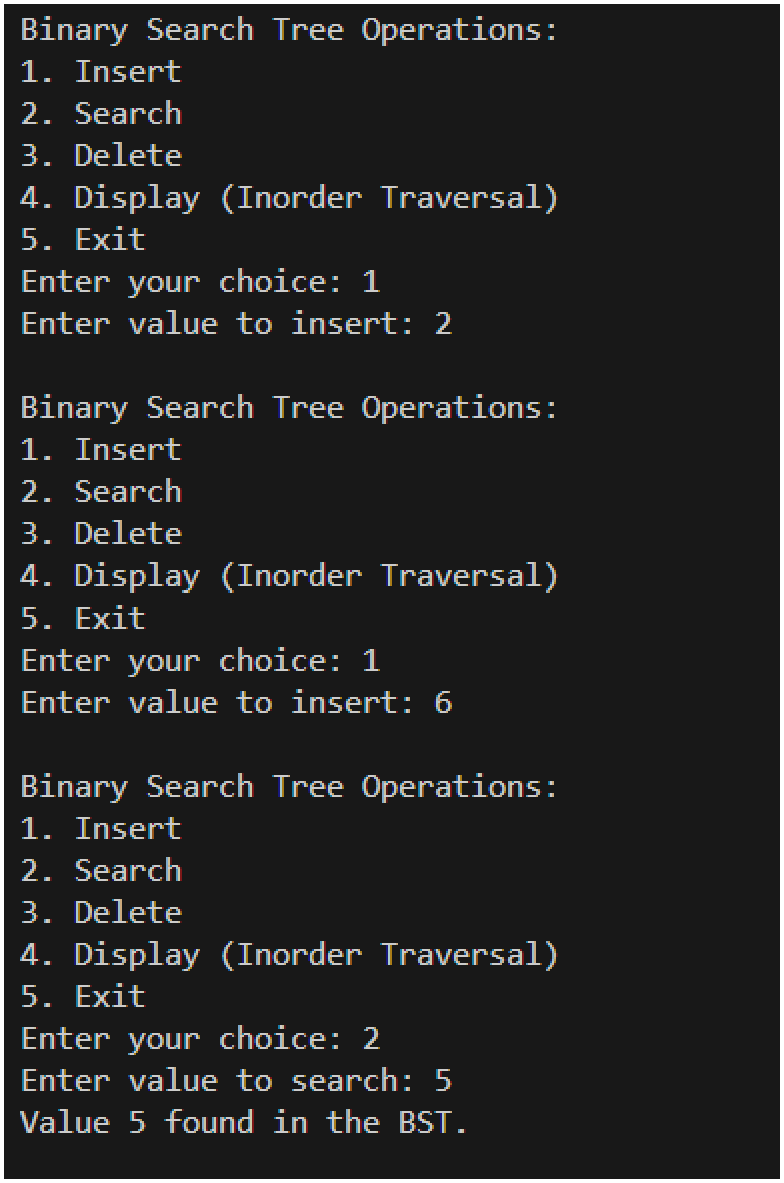


```c
            case 3:
                printf("Enter value to delete: ");
                scanf("%d", &value);
                root = deleteNode(root, value);
                break;
            case 4:
                printf("Inorder Traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

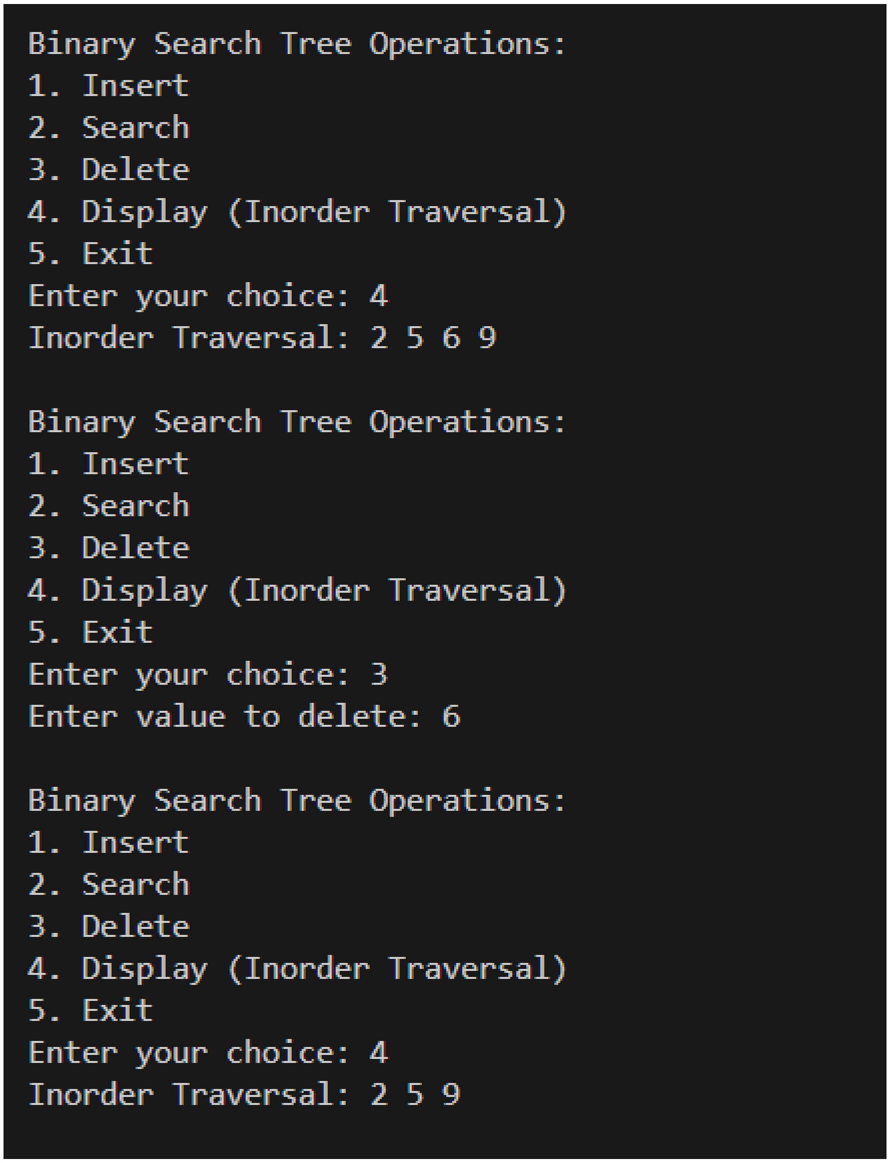
OUTPUT : --



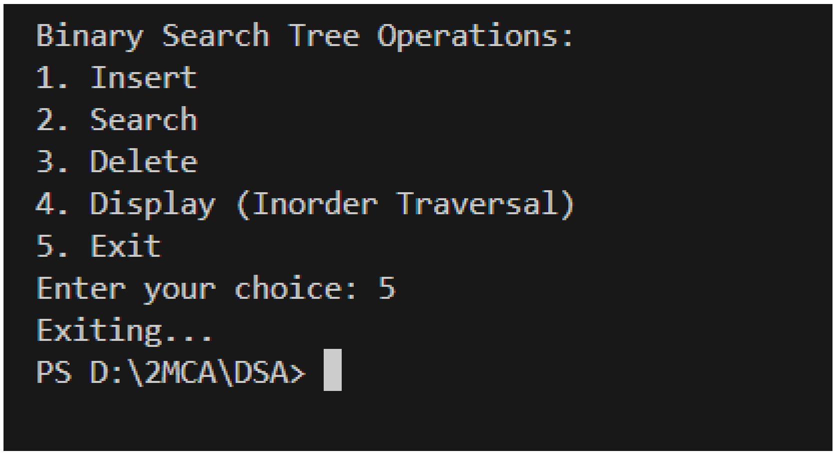
```
PS D:\2MCA\DSA> ./bst.exe

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

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

Binary Search Tree Operations:
1. Insert
2. Search
3. Delete
4. Display (Inorder Traversal)
5. Exit
Enter your choice: 1
Enter value to insert: 9
```



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

Binary Search Tree Operations:
1. Insert
2. Search
3. Delete
4. Display (Inorder Traversal)
5. Exit
Enter your choice: 1
Enter value to insert: 6

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



```
Binary Search Tree Operations:
1. Insert
2. Search
3. Delete
4. Display (Inorder Traversal)
5. Exit
Enter your choice: 4
Inorder Traversal: 2 5 6 9

Binary Search Tree Operations:
1. Insert
2. Search
3. Delete
4. Display (Inorder Traversal)
5. Exit
Enter your choice: 3
Enter value to delete: 6

Binary Search Tree Operations:
1. Insert
2. Search
3. Delete
4. Display (Inorder Traversal)
5. Exit
Enter your choice: 4
Inorder Traversal: 2 5 9
```



```
Binary Search Tree Operations:
1. Insert
2. Search
3. Delete
4. Display (Inorder Traversal)
5. Exit
Enter your choice: 5
Exiting...
PS D:\2MCA\DSA>
```