ADS LAB ASSINGMENT 4

Name-Sandesh Santosh Kadam
Class- SY IT-C Batch 1
PRN-12210695
Roll no-26

Q. Write C/C++ program to Create BST, Take input user choice, Perform Insertion and deletion on BST. (Include all cases for both)

Theory-

BST

- A binary search tree is a binary tree in which each node has at most two children, referred to as the left child and the right child.

- For any node in the tree, all the nodes in its left subtree have values less than the node's value, and all the nodes in its right subtree have values greater than the node's value.

- This property allows for efficient search, insertion, and deletion operations.

INSERTION

Insertion in a Binary Search Tree (BST) involves finding the appropriate position for the new node based on its key value while maintaining the BST property. If the tree is empty, the new node becomes the root. Otherwise, we traverse the tree to the left or right based on the key comparison until we find an empty spot.

DELETION

Deletion in a BST involves removing a node with a given key while ensuring that the BST property is preserved. There are three cases to consider: a node with no children, a node with one child, and a node with two children. In the first

case, we simply remove the node. In the second case, we bypass the node and link its child directly to its parent. In the third case, we find the node's in-order successor (smallest key in the right subtree), copy its key to the node to be deleted, and then delete the in-order successor node. This ensures that the BST property is maintained throughout the process.

Programming Lang. Used- C

Compiler Used- CodeBlocks

CODE-

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

struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

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

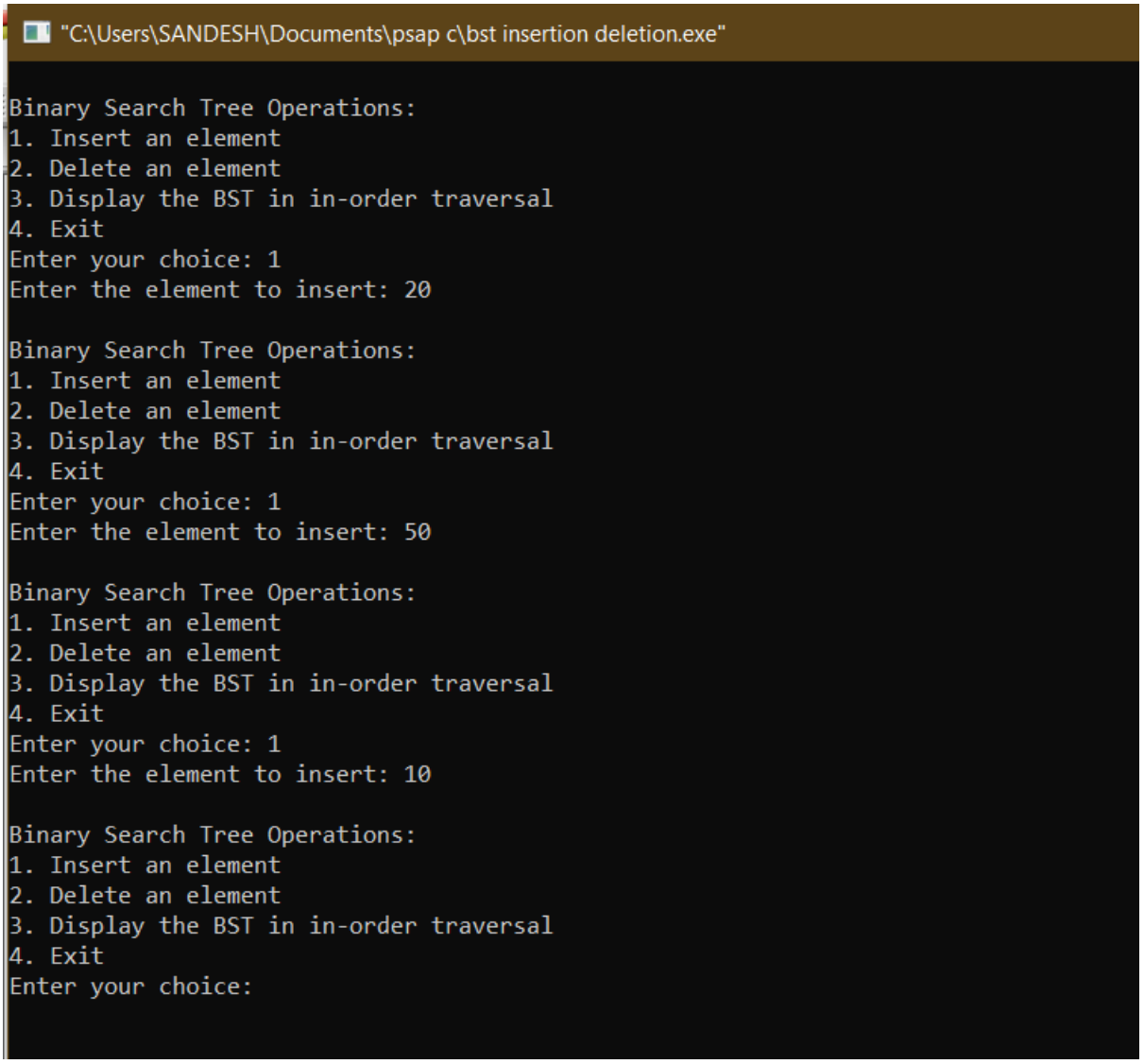
```
struct TreeNode* insert(struct TreeNode* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }

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

    return root;
}

struct TreeNode* findMin(struct TreeNode* node) {
    struct TreeNode* current = node;
    while (current && current->left != NULL) {
        current = current->left;
    }
    return current;
}

struct TreeNode* deleteNode(struct TreeNode* root, int
data) {
    if (root == NULL) {
        return root;
    }

    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
```

```
        root->right = deleteNode(root->right, data);
    } else {
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }

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

    return root;
}

void inOrderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}

int main() {
    struct TreeNode* root = NULL;
    int choice, data;
```

```
 do {
    printf("\nBinary Search Tree Operations:\n");
    printf("1. Insert an element\n");
    printf("2. Delete an element\n");
    printf("3. Display the BST in in-order traversal\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
       case 1:
          printf("Enter the element to insert: ");
          scanf("%d", &data);
          root = insert(root, data);
          break;
       case 2:

printf("Enter the element to delete: ");
          scanf("%d", &data);
          root = deleteNode(root, data);
          break;
       case 3:
          printf("BST in in-order traversal: ");
          inOrderTraversal(root);
          printf("\n");
          break;
       case 4:
          printf("Exiting...\n");
          break;
       default:
```

```
            printf("Invalid choice! Please try again.\n");
            break;
        }
    } while (choice != 4);

    return 0;
}
```

OUTPUT-



Inserted elements in the order 20 50 10

```
Binary Search Tree Operations:
1. Insert an element
2. Delete an element
3. Display the BST in in-order traversal
4. Exit
Enter your choice: 3
BST in in-order traversal: 10 20 50
```

BST in inorder traversal

```
Binary Search Tree Operations:
1. Insert an element
2. Delete an element
3. Display the BST in in-order traversal
4. Exit
Enter your choice: 2
Enter the element to delete: 20

Binary Search Tree Operations:
1. Insert an element
2. Delete an element
3. Display the BST in in-order traversal
4. Exit
Enter your choice: 3
BST in in-order traversal: 10 50
```

Deleted 20 and displayed BST