BHAVLEEN KAUR
102155010
3EIC-2

LAB ASSIGNMENT 8
Trees: BST and Traversing algorithms

1. Write a menu program to implement Binary Search Tree (Insertion, Deletion, traversing as In-order, Pre-order and Post-order).

```c
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int key;
    struct TreeNode *left, *right;
};
struct TreeNode* createNode(int key);
struct TreeNode* insert(struct TreeNode* root, int key);
struct TreeNode* deleteNode(struct TreeNode* root, int key);
void inorder(struct TreeNode* root);
void preorder(struct TreeNode* root);
void postorder(struct TreeNode* root);
int main() {
    struct TreeNode* root = NULL;
    int choice, key;
    do {
        printf("\nBinary Search Tree Operations Menu:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. In-order Traversal\n");
        printf("4. Pre-order Traversal\n");
        printf("5. Post-order Traversal\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &key);
                root = insert(root, key);
                break;
            case 2:
                printf("Enter value to delete: ");
                scanf("%d", &key);
                root = deleteNode(root, key);
                break;
            case 3:
                printf("In-order Traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 4:
```

BHAVLEEN KAUR
102155010
3EIC-2

```
43                  printf("Pre-order Traversal: ");
44                  preorder(root);
45                  printf("\n");
46                  break;
47              case 5:
48                  printf("Post-order Traversal: ");
49                  postorder(root);
50                  printf("\n");
51                  break;
52              case 6:
53                  printf("Exiting...\n");
54                  break;
55              default:
56                  printf("Invalid choice! Please try again.\n");
57          }
58      } while (choice != 6);
59
60      return 0;
61  }
62  struct TreeNode* createNode(int key) {
63      struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
            TreeNode));
64      newNode->key = key;
65      newNode->left = newNode->right = NULL;
66      return newNode;
67  }
68  struct TreeNode* insert(struct TreeNode* root, int key) {
69      if (root == NULL) {
70          return createNode(key);
71      }
72      if (key < root->key) {
73          root->left = insert(root->left, key);
74      } else if (key > root->key) {
75          root->right = insert(root->right, key);
76      }
77      return root;
78  }
79  struct TreeNode* minValueNode(struct TreeNode* node) {
80      struct TreeNode* current = node;
81      while (current && current->left != NULL) {
82          current = current->left;
```

BHAVLEEN KAUR
102155010
3EIC-2

```
    }
    return current;
}
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL) {
        return root;
    }
    if (key < root->key) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->key) {
        root->right = deleteNode(root->right, key);
    } 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 = minValueNode(root->right);
        root->key = temp->key;
        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}
void inorder(struct TreeNode* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}
void preorder(struct TreeNode* root) {
    if (root != NULL) {
        printf("%d ", root->key);
        preorder(root->left);
        preorder(root->right);
    }
}
void postorder(struct TreeNode* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->key);
    }
}
```

BHAVLEEN KAUR
102155010
3EIC-2

```
Binary Search Tree Operations Menu:
1. Insert
2. Delete
3. In-order Traversal
4. Pre-order Traversal
5. Post-order Traversal
6. Exit
Enter your choice: 1
Enter value to insert: 23
1
Binary Search Tree Operations Menu:
1. Insert
2. Delete
3. In-order Traversal
4. Pre-order Traversal
5. Post-order Traversal
6. Exit
Enter your choice: 1
Enter value to insert: 56

Binary Search Tree Operations Menu:
1. Insert
2. Delete
3. In-order Traversal
4. Pre-order Traversal
5. Post-order Traversal
6. Exit
Enter your choice:
1
Enter value to insert: 34

Binary Search Tree Operations Menu:
1. Insert
2. Delete
3. In-order Traversal
4. Pre-order Traversal
5. Post-order Traversal
6. Exit
Enter your choice: 3
In-order Traversal: 23 34 56
```

BHAVLEEN KAUR
102155010
3EIC-2

```
Binary Search Tree Operations Menu:
1. Insert
2. Delete
3. In-order Traversal
4. Pre-order Traversal
5. Post-order Traversal
6. Exit
Enter your choice: 4
Pre-order Traversal: 23 56 34

Binary Search Tree Operations Menu:
1. Insert
2. Delete
3. In-order Traversal
4. Pre-order Traversal
5. Post-order Traversal
6. Exit
Enter your choice: 5
Post-order Traversal: 34 56 23

Binary Search Tree Operations Menu:
1. Insert
2. Delete
3. In-order Traversal
4. Pre-order Traversal
5. Post-order Traversal
6. Exit
Enter your choice: 2
Enter value to delete: 56

Binary Search Tree Operations Menu:
1. Insert
2. Delete
3. In-order Traversal
4. Pre-order Traversal
5. Post-order Traversal
6. Exit
```