<u>**Experiment 9**</u>

CODE:

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

#define MAX_DATA_LENGTH 100

struct node {
   char data[MAX_DATA_LENGTH];
   struct node *left, *right;
};

int ncount, lcount;

int isValidData(const char *str) {
   if (str[0] == '\0') return 0;
   for (int i = 0; str[i] != '\0'; i++) {
      if (!isalnum(str[i])) return 0;
   }
   return 1;
}

void create(struct node **proot) {
   char str[MAX_DATA_LENGTH];

   printf("Enter the data value of node (-1 for no data or NULL): ");
   if (scanf("%s", str) != 1 || !isValidData(str)) {
      return;
   }
   if (strcmp(str, "-1") == 0) {
      return;
   }
```

```c
    *proot = (struct node *)malloc(sizeof(struct node));
    if (*proot == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    strcpy((*proot)->data, str);
    (*proot)->left = (*proot)->right = NULL;

    printf("\nEnter the left child of %s.\n", str);
    create(&((*proot)->left));

    printf("\nEnter the right child of %s.\n", str);
    create(&((*proot)->right));
}

void inorder(struct node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%s ", root->data);
        inorder(root->right);
    }
}

void preorder(struct node *root) {
    if (root != NULL) {
        printf("%s ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct node *root) {
    if (root != NULL) {
```

```
        postorder(root->left);
        postorder(root->right);
        printf("%s ", root->data);
    }
}

void nodecount(struct node *root) {
    if (root != NULL) {
        nodecount(root->left);
        ++ncount;
        nodecount(root->right);
    }
}

void leafnodecount(struct node *root) {
    if (root != NULL) {
        leafnodecount(root->left);
        if (root->left == NULL && root->right == NULL) {
            ++lcount;
            printf("%s ", root->data);
        }
        leafnodecount(root->right);
    }
}

int isBST(struct node *root, const char *min, const char *max) {
    if (root == NULL) return 1;

    if ((min != NULL && strcmp(root->data, min) <= 0) ||
        (max != NULL && strcmp(root->data, max) >= 0)) {
        return 0;
    }

    return isBST(root->left, min, root->data) && isBST(root->right, root-
>data, max);
```

```
}

void printTree(struct node *root, int level) {
    if (root == NULL) return;

    printTree(root->right, level + 1);

    for (int i = 0; i < level; i++) {
        printf("    ");
    }
    printf("%s\n", root->data);

    printTree(root->left, level + 1);
}

void displayBST(struct node *root) {
    if (isBST(root, NULL, NULL)) {
        printf("The tree is a Binary Search Tree. Displaying in hierarchical format:\n");
        printTree(root, 0);
        printf("\n");
    } else {
        printf("The tree is not a Binary Search Tree.\n");
    }
}

void insert(struct node **root, const char *data) {
    if (*root == NULL) {
        *root = (struct node *)malloc(sizeof(struct node));
        if (*root == NULL) {
            printf("Memory allocation failed.\n");
            return;
        }
        strcpy((*root)->data, data);
        (*root)->left = (*root)->right = NULL;
```

```
    } else {
        if (strcmp(data, (*root)->data) < 0) {
            insert(&(*root)->left, data);
        } else {
            insert(&(*root)->right, data);
        }
    }
}

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

void deleteNode(struct node **root, const char *data) {
    if (*root == NULL) return;

    if (strcmp(data, (*root)->data) < 0) {
        deleteNode(&(*root)->left, data);
    } else if (strcmp(data, (*root)->data) > 0) {
        deleteNode(&(*root)->right, data);
    } else {
        if ((*root)->left == NULL && (*root)->right == NULL) {
            free(*root);
            *root = NULL;
        } else if ((*root)->left == NULL) {
            struct node *temp = *root;
            *root = (*root)->right;
            free(temp);
        } else if ((*root)->right == NULL) {
            struct node *temp = *root;
            *root = (*root)->left;
            free(temp);
```

```
        } else {
            struct node *temp = findMin((*root)->right);
            strcpy((*root)->data, temp->data);
            deleteNode(&(*root)->right, temp->data);
        }
    }
}

int main() {
    struct node *root = NULL;
    int ch;

    printf("\033[2J\033[H");

    do {
        printf("1. Create\n");
        printf("2. Insert\n");
        printf("3. Delete\n");
        printf("4. Preorder\n");
        printf("5. Inorder\n");
        printf("6. Postorder\n");
        printf("7. Node Count\n");
        printf("8. Leaf Count\n");
        printf("9. Display BST\n");
        printf("10. Exit\n");

        printf("Enter your choice: ");
        if (scanf("%d", &ch) != 1) {
            printf("Invalid input. Exiting.\n");
            break;
        }

        switch (ch) {
            case 1:
                create(&root);
```

```
    break;

case 2: {
    char data[MAX_DATA_LENGTH];
    printf("Enter data to insert: ");
    scanf("%s", data);
    insert(&root, data);
    break;
}

case 3: {
    char data[MAX_DATA_LENGTH];
    printf("Enter data to delete: ");
    scanf("%s", data);
    deleteNode(&root, data);
    break;
}

case 4:
    preorder(root);
    printf("\n");
    break;

case 5:
    inorder(root);
    printf("\n");
    break;

case 6:
    postorder(root);
    printf("\n");
    break;

case 7:
    ncount = 0;
```

```
            nodecount(root);
            printf("No. of nodes present in the tree are %d\n", ncount);
            break;

        case 8:
            lcount = 0;
            leafnodecount(root);
            printf("No. of Leaf nodes present in the tree are %d\n",
lcount);
            break;

        case 9:
            displayBST(root);
            break;

        case 10:
            printf("Exiting...\n");
            break;

        default:
            printf("Invalid choice. Please try again.\n");
      }
   } while (ch != 10);

   return 0;
}
```

Output:

1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit

Enter your choice: 1
Enter the data value of node (-1 for no data or NULL): 50

Enter the left child of 50.
Enter the data value of node (-1 for no data or NULL): 30

Enter the left child of 30.
Enter the data value of node (-1 for no data or NULL): 20

Enter the left child of 20.
Enter the data value of node (-1 for no data or NULL): -1

Enter the right child of 20.
Enter the data value of node (-1 for no data or NULL): -1

Enter the right child of 30.
Enter the data value of node (-1 for no data or NULL): 40

Enter the left child of 40.
Enter the data value of node (-1 for no data or NULL): -1

Enter the right child of 40.
Enter the data value of node (-1 for no data or NULL): -1

Enter the right child of 50.
Enter the data value of node (-1 for no data or NULL): 70

Enter the left child of 70.
Enter the data value of node (-1 for no data or NULL): 60

Enter the left child of 60.
Enter the data value of node (-1 for no data or NULL): -1

Enter the right child of 60.
Enter the data value of node (-1 for no data or NULL): -1

Enter the right child of 70.
Enter the data value of node (-1 for no data or NULL): 80

Enter the left child of 80.
Enter the data value of node (-1 for no data or NULL): -1

Enter the right child of 80.
Enter the data value of node (-1 for no data or NULL): -1

1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit
Enter your choice: 4
50 30 20 40 70 60 80

1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit
Enter your choice: 5
20 30 40 50 60 70 80

1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit
Enter your choice: 6
20 40 30 60 80 70 50

1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST

```
10. Exit
Enter your choice: 7
No. of nodes present in the tree are 7

1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit
Enter your choice: 8
20 40 60 80 No. of Leaf nodes present in the tree are 4
1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit
Enter your choice: 9
The tree is a Binary Search Tree. Displaying in hierarchical format:
        80
   70
        60
50
        40
   30
        20
```

1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit
Enter your choice: 2
Enter data to insert: 25

1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit
Enter your choice: 2
Enter data to insert: 65

1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST

```
10. Exit
Enter your choice: 3
Enter data to delete: 40

1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit
Enter your choice: 3
Enter data to delete: 20
1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit
Enter your choice: 9
The tree is a Binary Search Tree. Displaying in hierarchical format:
          80
     70
                65
50        60


      30
           25
```

```
1. Create
2. Insert
3. Delete
4. Preorder
5. Inorder
6. Postorder
7. Node Count
8. Leaf Count
9. Display BST
10. Exit
Enter your choice: 10
Exiting...


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