**Assignment-4**

**Name of Student: Ayush Sanjay Dhangar**

**Batch: 02 Class: SY\_IT-A Roll No: 42**

**PRN: 12210406**

**Subject: IT2265 Advanced Data Structures**

**Problem Statement**: Write C or C++ program to create Binary search tree, insertion and searching and all deletion cases in BST

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

    int data;

    struct node \*right\_child;

    struct node \*left\_child;

};

struct node\* new\_node(int x) {

    struct node \*temp;

    temp = malloc(sizeof(struct node));

    temp->data = x;

    temp->left\_child = NULL;

    temp->right\_child = NULL;

    return temp;

}

struct node\* search(struct node \* root, int x) {

    if (root == NULL || root->data == x)

        return root;

    else if (x > root->data)

        return search(root->right\_child, x);

    else

        return search(root->left\_child, x);

}

struct node\* insert(struct node \* root, int x) {

    if (root == NULL)

        return new\_node(x);

    else if (x > root->data)

        root->right\_child = insert(root->right\_child, x);

    else

        root -> left\_child = insert(root->left\_child, x);

    return root;

}

struct node\* find\_minimum(struct node \* root) {

    if (root == NULL)

        return NULL;

    else if (root->left\_child != NULL)

        return find\_minimum(root->left\_child);

    return root;

}

struct node\* delete(struct node \* root, int x) {

    if (root == NULL)

        return NULL;

    if (x > root->data)

        root->right\_child = delete(root->right\_child, x);

    else if (x < root->data)

        root->left\_child = delete(root->left\_child, x);

    else {

        if (root->left\_child == NULL && root->right\_child == NULL) {

            free(root);

            return NULL;

        }

        else if (root->left\_child == NULL || root->right\_child == NULL) {

            struct node \*temp;

            if (root->left\_child == NULL)

                temp = root->right\_child;

            else

                temp = root->left\_child;

            free(root);

            return temp;

        }

        else {

            struct node \*temp = find\_minimum(root->right\_child);

            root->data = temp->data;

            root->right\_child = delete(root->right\_child, temp->data);

        }

    }

    return root;

}

void inorder(struct node \*root) {

    if (root != NULL) {

        inorder(root->left\_child);

        printf(" %d ", root->data);

        inorder(root->right\_child);

    }

}

int main() {

    struct node \*root = NULL;

    int choice, data;

    while (1) {

        printf("Binary Search Tree Operations: ");

        printf("\n1. Insert\n2. Delete\n3. Inorder Traversal\n4. Exit\nEnter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter the data to insert: ");

                scanf("%d", &data);

                root = insert(root, data);

                break;

            case 2:

                printf("Enter the data to delete: ");

                scanf("%d", &data);

                root = delete(root, data);

                break;

            case 3:

                printf("Inorder Traversal: ");

                inorder(root);

                printf("\n");

                break;

            case 4:

                exit(0);

            default:

                printf("Invalid choice!\n");

        }

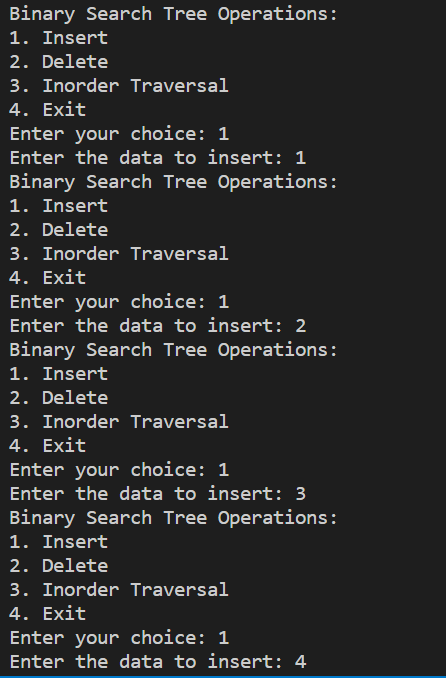
    }

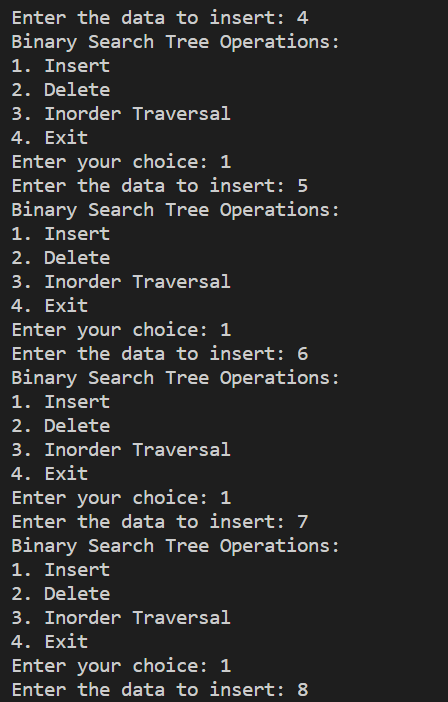
    return 0;

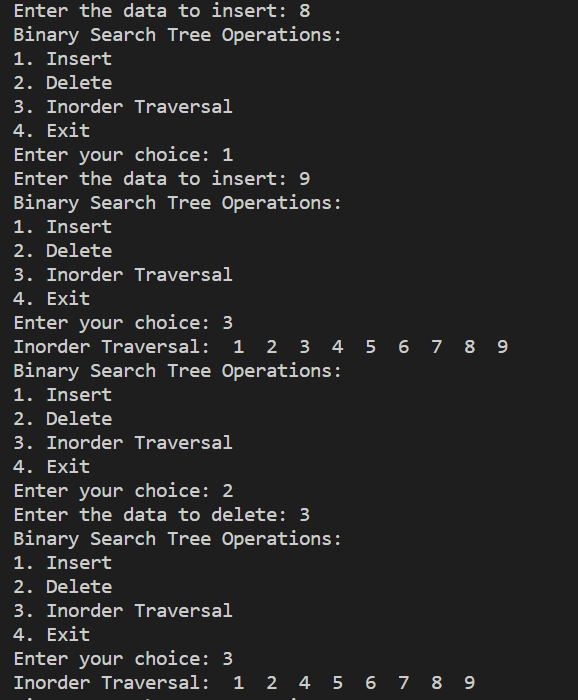
}

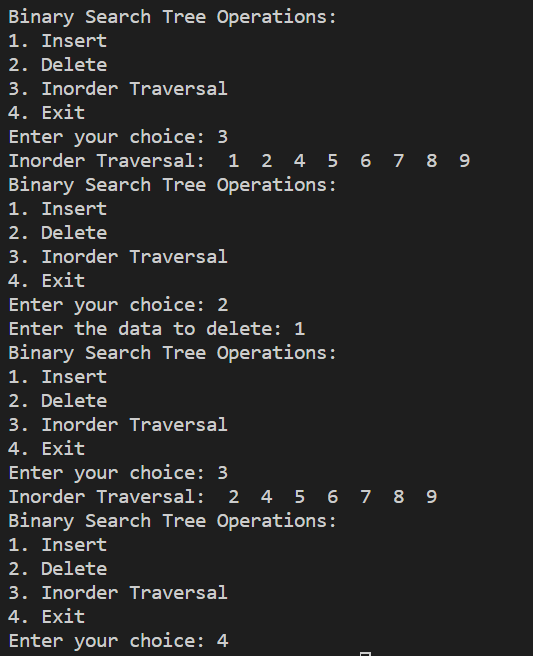
**Result:**

**Actual Output:**

****

****

****

****