**Problem Statement:**

Problem Statement: Menu-Driven Binary Search Tree (BST)

Design and implement a menu-driven program to perform various operations on a Binary Search Tree (BST). The program should allow the user to perform the following operations:

1. Insert a node: The user should be able to insert a new node into the BST by providing a value.

2. Delete a node: The user should be able to delete a node from the BST by providing a value.

3. Search for a node: The user should be able to search for a specific value in the BST and determine whether it exists or not.

4. Find the minimum value: The program should find and display the minimum value present in the BST.

5. Find the maximum value: The program should find and display the maximum value present in the BST.

6. Inorder traversal: The program should perform an inorder traversal of the BST and display the nodes in sorted order.

7. Preorder traversal: The program should perform a preorder traversal of the BST and display the nodes.

8. Postorder traversal: The program should perform a postorder traversal of the BST and display the nodes.

9. Display the BST: The program should display the structure of the BST in a visually appealing format.

10. Exit the program: The user should be able to exit the program.

The program should display a menu to the user with the above options and allow them to select an operation by entering the corresponding option number. After performing the requested operation, the program should return to the menu for further operations until the user chooses to exit.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the BST

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new node

struct Node\* createNode(int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

printf("Memory error!\n");

return NULL;

}

newNode->data = value;

newNode->left = newNode->right = NULL;

return newNode;

}

// Function to insert a node into the BST

struct Node\* insertNode(struct Node\* root, int value) {

if (root == NULL) {

return createNode(value);

}

if (value < root->data) {

root->left = insertNode(root->left, value);

} else if (value > root->data) {

root->right = insertNode(root->right, value);

}

return root;

}

// Function to search for an element in the BST

struct Node\* searchNode(struct Node\* root, int value) {

if (root == NULL || root->data == value) {

return root;

}

if (value < root->data) {

return searchNode(root->left, value);

}

return searchNode(root->right, value);

}

// Function to find the minimum value in a BST

struct Node\* minValueNode(struct Node\* node) {

struct Node\* current = node;

while (current && current->left != NULL) {

current = current->left;

}

return current;

}

// Function to delete a node from the BST

struct Node\* deleteNode(struct Node\* root, int value) {

if (root == NULL) {

return root;

}

if (value < root->data) {

root->left = deleteNode(root->left, value);

} else if (value > root->data) {

root->right = deleteNode(root->right, value);

} else {

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;

}

struct Node\* temp = minValueNode(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

// Function to perform inorder traversal of the BST

void inorderTraversal(struct Node\* root) {

if (root == NULL) {

return;

}

inorderTraversal(root->left);

printf("%d ", root->data);

inorderTraversal(root->right);

}

// Function to perform preorder traversal of the BST

void preorderTraversal(struct Node\* root) {

if (root == NULL) {

return;

}

printf("%d ", root->data);

preorderTraversal(root->left);

preorderTraversal(root->right);

}

// Function to perform postorder traversal of the BST

void postorderTraversal(struct Node\* root) {

if (root == NULL) {

return;

}

postorderTraversal(root->left);

postorderTraversal(root->right);

printf("%d ", root->data);

}

int main() {

printf("Yogesh Pal Parmar MCA 2B 70\n");

struct Node\* root = NULL;

int choice, value;

while (1) {

printf("BST Menu:\n");

printf("1. Insert Element\n");

printf("2. Search Element\n");

printf("3. Delete Element\n");

printf("4. Inorder Traversal\n");

printf("5. Preorder Traversal\n");

printf("6. Postorder Traversal\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to insert: ");

scanf("%d", &value);

root = insertNode(root, value);

break;

case 2:

printf("Enter element to search: ");

scanf("%d", &value);

if (searchNode(root, value) != NULL) {

printf("Element found!\n");

} else {

printf("Element not found!\n");

}

break;

case 3:

printf("Enter element to delete: ");

scanf("%d", &value);

root = deleteNode(root, value);

break;

case 4:

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\n");

break;

case 5:

printf("Preorder Traversal: ");

preorderTraversal(root);

printf("\n");

break;

case 6:

printf("Postorder Traversal: ");

postorderTraversal(root);

printf("\n");

break;

case 7:

exit(0);

break;

default:

printf("Invalid choice!\n");

}

}

return 0;

}

**Output:**



