# IMPLEMENTATION OF BINARY SEARCH TREE

# Aim:

To implement a binary search tree (BST) with insertion, deletion, search, and in-order traversal operations using C programming language.

# Algorithm:

1. Start.  
2. Define the structure of the tree node.  
3. Implement a function to create a new node.  
4. Implement a function to insert a node into the BST.  
5. Implement a function to find the minimum value node in the BST.  
6. Implement a function to delete a node from the BST.  
7. Implement a function to perform in-order traversal of the BST.  
8. Implement a function to search for a value in the BST.  
9. Create a main function to provide a menu-driven interface for inserting, deleting, searching, and displaying the BST.  
10. End.

# Program:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node\* left;  
 struct Node\* right;  
};  
  
struct Node\* createNode(int data) {  
 struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->left = NULL;  
 newNode->right = NULL;  
 return newNode;  
}

struct Node\* insertNode(struct Node\* root, int data) {  
 if (root == NULL) {  
 root = createNode(data);  
 } else if (data < root->data) {  
 root->left = insertNode(root->left, data);  
 } else {  
 root->right = insertNode(root->right, data);  
 }  
 return root;  
}  
  
struct Node\* findMin(struct Node\* root) {  
 while (root->left != NULL) root = root->left;  
 return root;  
}  
  
struct Node\* deleteNode(struct Node\* root, int data) {  
 if (root == NULL) {  
 printf("Element not present in tree.\n");  
 return root;  
 } else 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 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 = findMin(root->right);  
 root->data = temp->data;  
 root->right = deleteNode(root->right, temp->data);  
 }  
 return root;  
}  
  
void inorderTraversal(struct Node\* root) {  
 if (root != NULL) {  
 inorderTraversal(root->left);  
 printf("%d ", root->data);  
 inorderTraversal(root->right);  
 }  
}  
  
struct Node\* search(struct Node\* root, int data) {  
 if (root == NULL || root->data == data) {  
 return root;  
 } else if (data < root->data) {  
 return search(root->left, data);  
 } else {  
 return search(root->right, data);  
 }  
}  
  
int main() {  
 struct Node\* root = NULL;  
 int choice, value;  
 do {  
 printf("\nMenu:\n");  
 printf("1. Insert\n");  
 printf("2. Delete\n");  
 printf("3. Search\n");  
 printf("4. Display\n");  
 printf("5. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 printf("Enter value to insert: ");  
 scanf("%d", &value);  
 root = insertNode(root, value);  
 break;  
 case 2:  
 printf("Enter value to delete: ");  
 scanf("%d", &value);  
 root = deleteNode(root, value);  
 break;  
 case 3:  
 printf("Enter value to search: ");  
 scanf("%d", &value);  
 if (search(root, value) != NULL) {  
 printf("Node with value %d found in the binary search tree.\n", value);  
 } else {  
 printf("Node with value %d not found in the binary search tree.\n", value);  
 }  
 break;  
 case 4:  
 printf("In-order traversal of the binary search tree:\n");  
 if (root == NULL) {  
 printf("Tree is empty.\n");  
 } else {  
 inorderTraversal(root);  
 printf("\n");  
 }  
 break;  
 case 5:  
 printf("Exiting...\n");  
 break;  
 default:  
 printf("Invalid choice. Please try again.\n");  
 }  
 } while (choice != 5);  
  
 return 0;  
}

# Output:

Menu:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 1

Enter value to insert: 10

Menu:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 1

Enter value to insert: 5

Menu:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 1

Enter value to insert: 15

Menu:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 1

Enter value to insert: 2

Menu:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 4

In-order traversal of the binary search tree:

2 5 10 15

Menu:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 3

Enter value to search: 10

Node with value 10 found in the binary search tree.

Menu:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 2

Enter value to delete: 5

Menu:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 4

In-order traversal of the binary search tree:

2 10 15

Menu:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 5

Exiting......

# Result:

The program successfully implemented a binary search tree with operations to insert, delete, search, and display nodes using in-order traversal.