Experiment No.9
Implement Binary Search Tree ADT using Linked List.
Name: Shravani Sandeep Raut
Roll No: 48
Date of Performance:
Date of Submission:

Experiment No. 9: Binary Search Tree Operations

Aim : Implementation of Binary Search Tree ADT using Linked List.

Objective:

1) Understand how to implement a BST using a predefined BST ADT.

2) Understand the method of counting the number of nodes of a binary tree.

Theory:

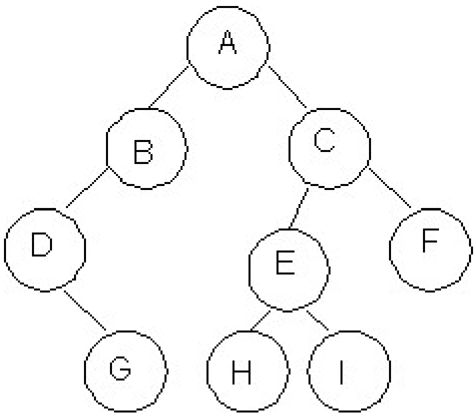
A binary tree is a finite set of elements that is either empty or partitioned into disjoint subsets. In other words nodes in a binary tree have at most two children and each child node is referred to as left or right child.

Traversals in trees can be in one of the three ways: preorder, postorder, inorder.

Preorder Traversal

Here the following strategy is followed in sequence

1. Visit the root node R
2. Traverse the left subtree of R
3. Traverse the right subtree of R



Description	**Output**
Visit Root	A
Traverse left sub tree – step to B then D	ABD
Traverse right subtree – step to G	ABDG
As left subtree is over. Visit root , which is already visited so go for right subtree	ABDGC
Traverse the left subtree	ABDGCEH
Traverse the right sub tree	ABDGCEHIF

Inorder Traversal

Here the following strategy is followed in sequence

1. Traverse the left subtree of R
2. Visit the root node R
3. Traverse the right sub tree of R

Description	Output
Start with root and traverse left sub tree from A-B-D	D
As D doesn't have left child visit D and go for right subtree of D which is G so visit this.	DG
Backtrack to D and then to B and visit it.	DGB
Backtrack to A and visit it	DGBA
Start with right sub tree from C-E-H and visit H	DGBAH
Now traverse through parent of H which is E and then I	DGBAHEI
Backtrack to C and visit it and then right subtree of E which is F	DGBAHEICF

Postorder Traversal

Here the following strategy is followed in sequence

1. Traverse the left subtree of R
2. Traverse the right sub tree of R
3. Visit the root node R

Description	Output
Start with left sub tree from A-B-D and then traverse right sub tree to get G	G
Now Backtrack to D and visit it then to B and visit it.	GD
Now as the left sub tree is over go for right sub tree	GDB
In right sub tree start with leftmost child to visit H followed by I	GDBHI
Visit its root as E and then go for right sibling of C as F	GDBHIEF
Traverse its root as C	GDBHIEFC
Finally a root of tree as A	GDBHIEFCA

Algorithm

Algorithm: PREORDER(ROOT)

Algorithm :

Function Pre-order(root)

- Start
- If root is not null then

Display the data in root

Call pre order with left pointer of root(root -> left)

Call pre order with right pointer of root(root -> right)

- Stop

Algorithm: INORDER(ROOT)

Algorithm :

Function in-order(root)

- Start
- If root is not null then

Call in order with left pointer of root (root -> left)

Display the data in root

Call in order with right pointer of root(root -> right)

- Stop

Algorithm: POSTORDER(ROOT)

Algorithm :

Function post-order (root)

- Start
- If root is not null then

Call post order with left pointer of root (root -> left)

Call post order with right pointer of root (root -> right)

Display the data in root

- Stop

Code:

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

// Define the structure for the Binary Search Tree node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
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;
}

// Function to insert a node in the Binary Search Tree
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

// Preorder Traversal (Root -> Left -> Right)
void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

// Inorder Traversal (Left -> Root -> Right)
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

// Postorder Traversal (Left -> Right -> Root)
void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
```

```
    }
}

// Function to count the number of nodes in the BST
int countNodes(struct Node* root) {
    if (root == NULL) {
        return 0;
    }
    return 1 + countNodes(root->left) + countNodes(root->right);
}

// Main Function
int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

    printf("Preorder Traversal: ");
    preorder(root);
    printf("\n");

    printf("Inorder Traversal: ");
    inorder(root);
    printf("\n");

    printf("Postorder Traversal: ");
    postorder(root);
    printf("\n");

    printf("Total number of nodes: %d\n", countNodes(root));

    return 0;
}
```

Output:

```
Preorder Traversal: 50 30 20 40 70 60 80
Inorder Traversal: 20 30 40 50 60 70 80
Postorder Traversal: 20 40 30 60 80 70 50
Total number of nodes: 7
```

Conclusion:

1)Write a function in C program to count the number of nodes in a binary search tree?

To count the number of nodes in a Binary Search Tree (BST) in C, you can use a recursive function that traverses the tree and counts each node as it visits.

```
// Function to count the number of nodes in the BST
int countNodes(struct Node* root)
{
        if (root == NULL)
        {
                return 0;
        }
        return 1 + countNodes(root->left) + countNodes(root->right);
}
```