| **Experiment No.9** |
|---|
| Implement Binary Search Tree ADT using Linked List. |
| Name: Sharvari Anand Bhondekar |
| Roll No: 06 |
| Date of Performance: |
| Date of Submission: |
| Marks: |
| Sign: |

**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.

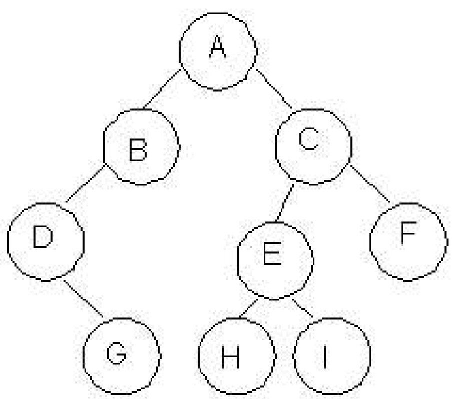
**Theory:**

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 of a 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;
}

// Pre-order traversal: Root -> Left -> Right
void preOrder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);      // Display the data in root
        preOrder(root->left);          // Traverse left subtree
        preOrder(root->right);         // Traverse right subtree
    }
}

// In-order traversal: Left -> Root -> Right
void inOrder(struct Node* root) {
    if (root != NULL) {
        inOrder(root->left);           // Traverse left subtree
        printf("%d ", root->data);      // Display the data in root
```

```
        inOrder(root->right);           // Traverse right subtree
    }
}

// Post-order traversal: Left -> Right -> Root
void postOrder(struct Node* root) {
    if (root != NULL) {
        postOrder(root->left);          // Traverse left subtree
        postOrder(root->right);         // Traverse right subtree
        printf("%d ", root->data);      // Display the data in root
    }
}

int main() {
    // Manually creating a binary tree
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);
    root->right->right = createNode(7);

    // Display the tree traversals
    printf("Pre-order traversal: ");
    preOrder(root);         // Call pre-order traversal
```

```
    printf("\n");

    printf("In-order traversal: ");
    inOrder(root);          // Call in-order traversal
    printf("\n");

    printf("Post-order traversal: ");
    postOrder(root);        // Call post-order traversal
    printf("\n");

    return 0;
}
```

**Output:**

```
Pre-order traversal: 1 2 4 5 3 6 7
In-order traversal: 4 2 5 1 6 3 7
Post-order traversal: 4 5 2 6 7 3 1
```

**Conclusion:**

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

**code:**

```
#include <stdio.h>

#include <stdlib.h>

// Define the structure of a 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 count the number of nodes in a Binary Search Tree
int countNodes(struct Node* root) {
    if (root == NULL) {
        return 0;  // Base case: If tree is empty, count is 0
    }

    // Recursive case: 1 (for the root) + count of left subtree + count of right subtree
    return 1 + countNodes(root->left) + countNodes(root->right);
}
```

```
int main() {
    // Manually creating a binary search tree
    struct Node* root = createNode(10);
    root->left = createNode(5);
    root->right = createNode(20);
    root->left->left = createNode(3);
    root->left->right = createNode(7);
    root->right->left = createNode(15);
    root->right->right = createNode(25);

    // Counting the number of nodes in the tree
    int totalNodes = countNodes(root);

    // Printing the total number of nodes
    printf("Total number of nodes in the binary search tree: %d\n", totalNodes);

    return 0;
}
```

**OUTPUT:**

```
Total number of nodes in the binary search tree: 7


...Program finished with exit code 0
Press ENTER to exit console.
```