K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: D2 Roll No.: 16010122323

Experiment / assignment / tutorial No. 8

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of BST & Binary tree traversal techniques.

Objective: To Understand and Implement Binary Search Tree, Preorder, Postorder and Inorder Traversal Techniques.

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4. https://www.geeksforgeeks.org/binary-tree-data-structure/
5. https://www.thecrazyprogrammer.com/2015/03/c-program-for-binary-search-tree-insertion.html

Abstract:

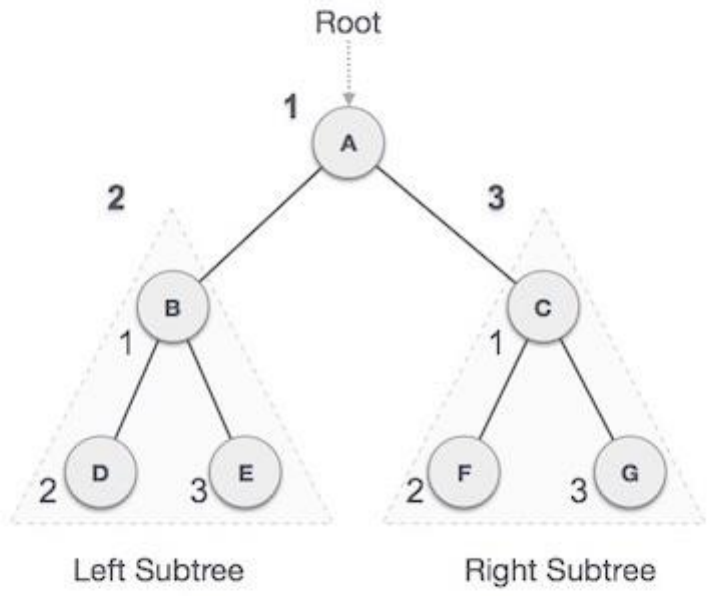
A tree is a non- linear data structure used to represent hierarchical relationship existing among several data items. It is a finite set of one or more data items such that, there is a special data item called the root of the tree. Its remaining data items are partitioned into number of mutually exclusive subsets, each of which is itself a tree, and they are called subtrees.

A binary tree is a finite set of nodes. It is either empty or It consists a node called root with two disjoint binary trees-Left subtree, Right subtree. The Maximum degree of any node is 2

A Binary Search Tree is a node-based binary tree data structure in which the left subtree of a node contains only nodes with keys lesser than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. The left and right subtree each must also be a binary search tree.

Related Theory: -
Preorder Traversal of BST

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.



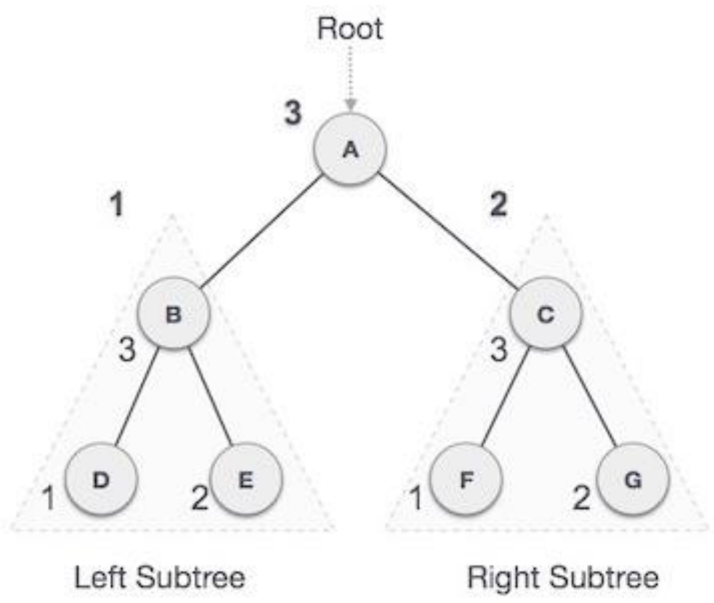
We start from A, and following pre-order traversal, we first visit A itself and then move to its left subtree B. B is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be −

$$A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$$

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on an expression tree.

Postorder Traversal of BST

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.



We start from A, and following Post-order traversal, we first visit the left subtree B. B is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be −

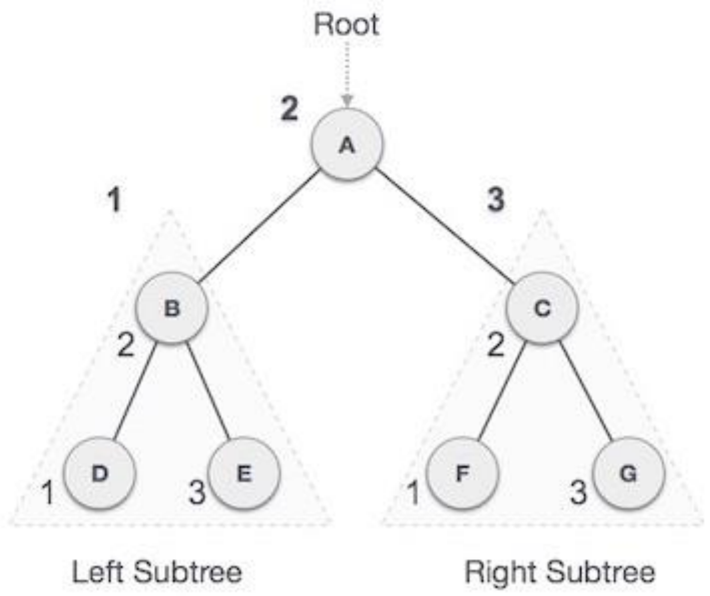
$$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$$

Postorder traversal is used to delete the tree. Postorder traversal is also useful to get the postfix expression of an expression tree.

Inorder Traversal of BST

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.

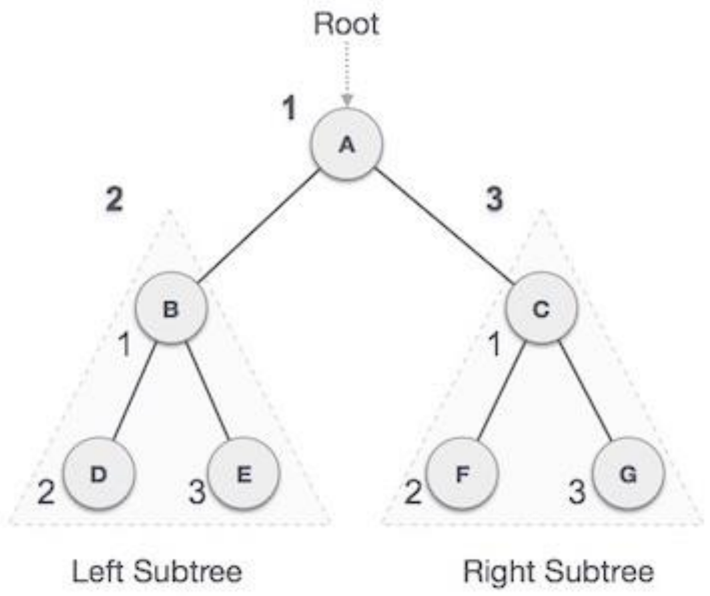


We start from A, and following in-order traversal, we move to its left subtree B. B is also traversed in-order. The process goes on until all the nodes are visited. The output of inorder traversal of this tree will be −

$$D \to B \to E \to A \to F \to C \to G$$

Diagram for :

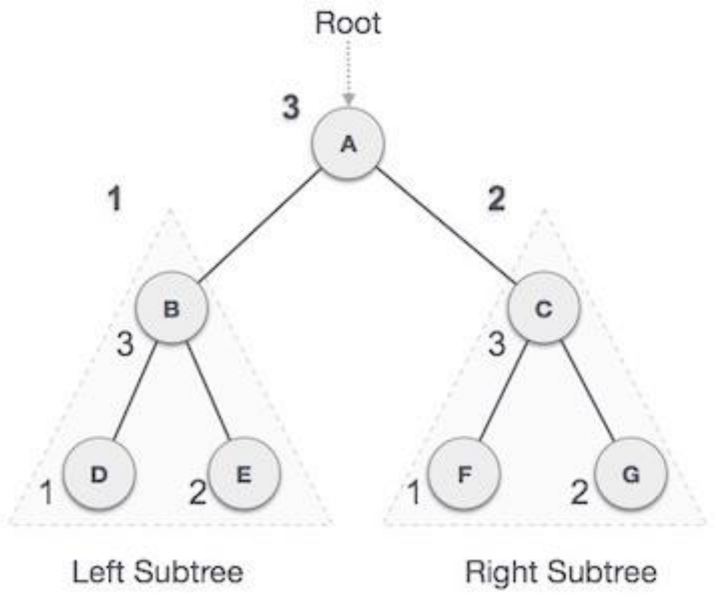
Preorder Traversal of BST



The output of pre-order traversal of this tree will be –

A → B → D → E → C → F → G

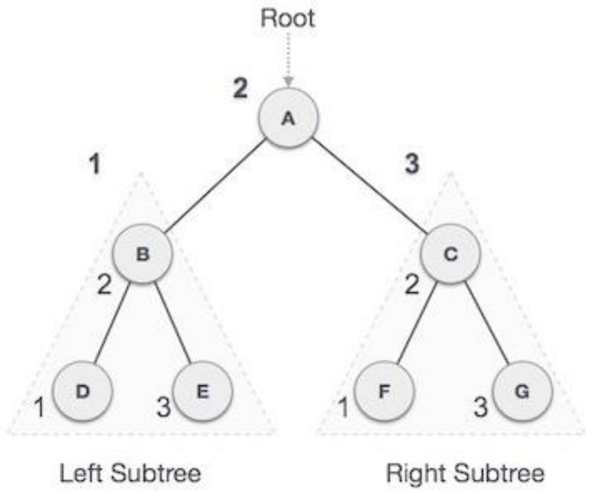
Postorder Traversal of BST



The output of post-order traversal of this tree will be –

D → E → B → F → G → C → A

Inorder Traversal of BST



The output of inorder traversal of this tree will be −

$$D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$$

Algorithm for Implementation of BST & Binary tree traversal techniques:

1. **Preorder Traversal**

```
Until all nodes are traversed −
Step 1 − Visit root node.
Step 2 − Recursively traverse left subtree.
Step 3 − Recursively traverse right subtree.
```

2. **Postorder Traversal**

```
Until all nodes are traversed –
Step 1 – Recursively traverse left subtree.
Step 2 – Recursively traverse right subtree.
Step 3 – Visit root node.
```

3. **Inorder Traversal**

```
Until all nodes are traversed –
Step 1 – Recursively traverse left subtree.
Step 2 – Visit root node.
Step 3 – Recursively traverse right subtree.
```

Implementation Details:

1) Enlist all the Steps followed, and various options explored.

- We made a switch case to implement a menu driven program. The options available are add (make a bst), search(indicate level at which node is present) and traverse (inorder, postorder, preorder)
- All the functions written are recursive as multiple branches are present.
- for add, we compare the value of term to be inserted with node. If it's lesser, we traverse towards left, else right. We continue the traversal(i.e. pass the value back recursively) in the appropriate directions till we node whose required subtree is null and attach our node there.
- For search, we again go on comparing the value of the term to be searched with node. If it's lesser, we traverse towards left, else right. We do the traversal using recursion. For each time we go right or left we increment the value of variable pos that was initially 0. When we reach the required value we print it out along with pos that indicates the depth at which we found it.

If we reach a leaf without encountering the value we print out that the value was not present in the tree.

- For traversal, we use recursion, For inorder we first go left, then print the existing node and then go right, Since the left statement is first in the order we always go on traversing to the left. We can't traverse further left, we print out the node. When the 1st two steps can’t be executed, we execute the 3rd step and go right.
- Postorder and preorder are executed similarly, with the only difference being in the order of execution of steps. In post order, the order is: go left-go right-print node. In preorder, the order is: print node-go left- go right.
- if not switch case we could have implemented an if-else.
- for search we could have just printed out if the element is present.
- Alternatively, we could have given the number of times we went left or right.

Assumptions made for Input:

- All terms entered are distinct.
- IDE is functioning properly
- rt denoted the right and left denotes the left child of a node
- input is in form of integers

Built-In Functions Used:

1)printf

2)scanf

3)if-else

4)switch

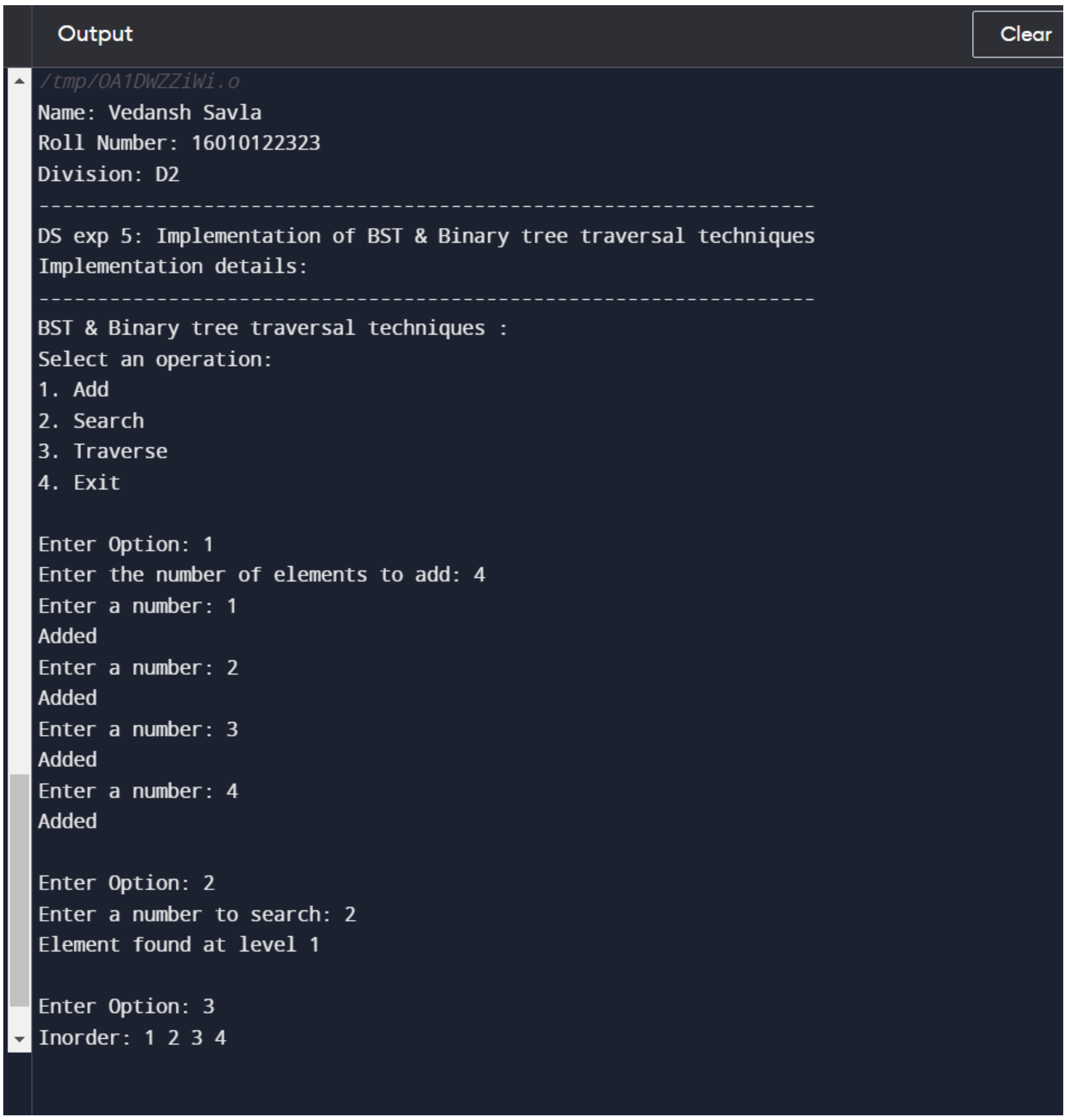
5)break

Program source code for Implementation of BST & Binary tree traversal techniques

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

struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node *createNode(int info);
struct Node *insert(struct Node *root, int info);
void search(struct Node *root, int info, int level);
void inorder(struct Node *root);
void preorder(struct Node *root);
void postorder(struct Node *root);

struct Node *root = NULL;

struct Node *createNode(int info) {
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node));
    temp->data = info;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}

struct Node *insert(struct Node *root, int info) {
    if (root == NULL) {
        return createNode(info);
    } else if (info < root->data) {
        root->left = insert(root->left, info);
    } else if (info > root->data) {
        root->right = insert(root->right, info);
    }
    return root;
}

void search(struct Node *root, int info, int level) {
    if (root == NULL) {
        printf("Element not found\n");
    } else if (info == root->data) {
        printf("Element found at level %d\n", level);
    } else if (info < root->data) {
        search(root->left, info, level + 1);
    } else {
        search(root->right, info, level + 1);
    }
}

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

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

void postorder(struct Node *root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

int main(void) {
    int choice, num, n;

    printf("Name: Vedansh Savla\n");
    printf("Roll Number: 16010122323\nDivision: D2\n");
    printf("-------------------------------------------------------------
-----\n");
    printf("DS exp 5: Implementation of BST & Binary tree traversal
techniques\n");
    printf("Implementation details:\n");
    printf("-------------------------------------------------------------
-----\n");
```

```
    printf("BST & Binary tree traversal techniques :\n");

    printf("Select an operation:\n");
    printf("1. Add\n2. Search\n3. Traverse\n4. Exit\n");

    while (1) {
        printf("\nEnter Option: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the number of elements to add: ");
                int count;
                scanf("%d", &count);
                for (int i = 0; i < count; i++) {
                    printf("Enter a number: ");
                    scanf("%d", &num);
                    root = insert(root, num);
                    printf("Added\n");
                }
                break;
            case 2:
                printf("Enter a number to search: ");
                scanf("%d", &n);
                search(root, n, 0);
                break;
            case 3:
                printf("\nInorder: ");
                inorder(root);
                printf("\nPostorder: ");
                postorder(root);
                printf("\nPreorder: ");
                preorder(root);
                break;
            case 4:
                printf("Exiting\n");
                exit(0);
            default:
                printf("Invalid option!!! Please choose again\n");
        }
    }

    return 0;
}
```

Output:



```
/tmp/OA1DWZZiWi.o
Name: Vedansh Savla
Roll Number: 16010122323
Division: D2
-------------------------------------------------------------------
DS exp 5: Implementation of BST & Binary tree traversal techniques
Implementation details:
-------------------------------------------------------------------
BST & Binary tree traversal techniques :
Select an operation:
1. Add
2. Search
3. Traverse
4. Exit

Enter Option: 1
Enter the number of elements to add: 4
Enter a number: 1
Added
Enter a number: 2
Added
Enter a number: 3
Added
Enter a number: 4
Added

Enter Option: 2
Enter a number to search: 2
Element found at level 1

Enter Option: 3
Inorder: 1 2 3 4
```

Explain the Importance of the approach followed by you

.

- use of switch case makes code cleaner
- dividing the code into functions makes it easier to understand
- recursive functions make it easier to deal with branches

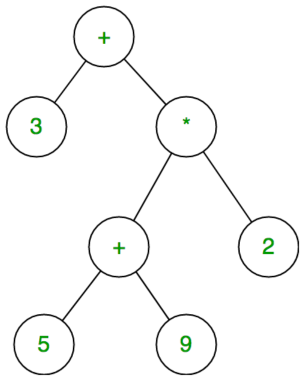
- using an variable to keep track of loops makes the search operation more precise and tells us the level of a bst the element would appear in

Conclusion:-
Thus, we implemented the operations on BST successfully.

PostLab Questions:

1. **Illustrate 2 Applications of Trees.**

1. storing of arithmetic operations in expression trees.
They can be stored such that the children represent the operands and the node represents the operations.
Following is an expression tree for 3 + ((5+9)*2)



2. For morse code implementation
morse code(or dot and dash) can be decoded and encoded more easily with the help of a binary tree. This prevents us from visiting the chart and searching for each letter again and again. Morse code is used to send secret messages using sounds like beep beep beeeep (dot dot dash). The left subtree is accessed using dots and the left ones using dashes. For example, using the following subtree we can say that A is .- and U is ..- and so on.

2. **Compare and Contrast between B Tree and B+ Tree?**

B tree	**B+ tree**

Data is stored in internal nodes as well as leaves	Data is only stored in leaves
Deletion operation is complex for internal node	Deletion operation is easier as all data is found in leaves
B tree has no redundant value.	B+ trees store redundant search key
Duplicated keys are not maintained	Duplicate keys are maintained
Leaf nodes are not stored as a structural linked list.	Leaf nodes are stored as a structural linked list.