Batch: A_1 Roll No.: 16010123012

Experiment / assignment / tutorial No. 7

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 along with Insertion, Deletion and 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: -
Algorithm: Preorder Traversal of BST-

Visit the root node.
Traverse the left subtree by recursively performing a preorder traversal.
Traverse the right subtree by recursively performing a preorder traversal.

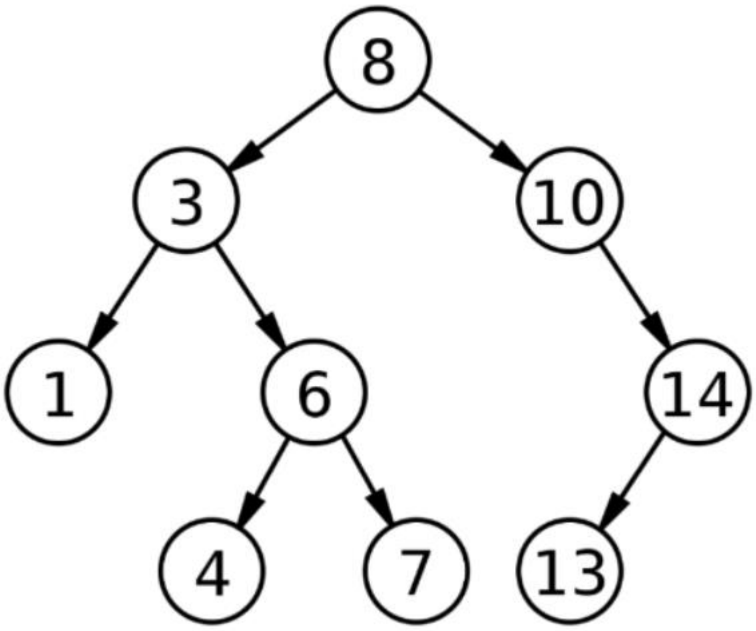
Algorithm: Postorder Traversal of BST-

Traverse the left subtree by recursively performing a postorder traversal.
Traverse the right subtree by recursively performing a postorder traversal.
Visit the root node.

Algorithm: Inorder Traversal of BST-

Traverse the left subtree by recursively performing an inorder traversal.
Visit the root node.
Traverse the right subtree by recursively performing an inorder traversal.

An example BST :



Preorder Traversal:

8 3 1 6 4 7 10 14 13

Postorder Traversal:

1 4 7 6 3 13 14 10 8

Inorder Traversal:

1 3 4 6 7 8 10 13 14

Algorithm for Implementation of BST:

Search in BST: -
1. Start from the root.

2. Compare the searching element with root, if less than root, then recurse for left, else recurse for right.

3. If the element to search is found anywhere, return true, else return false.

Insertion in BST: -

1. Start from the root.

2. Compare the inserting element with root, if less than root, then recurse for left, else recurse for right.

3. After reaching the end, just insert that node at left (if less than current) else right.

Deletion in BST: -

1. Search the node that to be deleted in BST.

2. If node has left or right subtree then find the inorder predecessor or inorder successor respectively (call it new node) and replace the node data value with its data. Then make recursive call to the delete function for that new node.

3 Else delete the node and assign the parent's child pointer(which is node) to NULL.

Implementation Details:

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

A structure called Node is created which has two struct pointers (left and right) and 1 int variable (data).

A class called BST is created with functions for insertion, deletion and searching. For insertion in BST: -
• If root is null create new node and assign the data to it.
• Else traverse in tree according to the data of node.

For searching: -

• If node is null then the node with required data isn't there in the BST.

• Else traverse in tree according to the data of node.

For deletion: -

• Search the node to be deleted.

• If the node has left or right child then replace its data value with inorder

successor (if right child is present) or predecessor of current node. Then search of

that node in the corresponding subtree.

• Else delete the node and assign its parents pointer (which points to current node) as NULL.

An object of BST class is created and a menu driven program is made for the user to choose an operation.

Assumptions made for Input:

The value of all nodes are integers and |value|<=INTMAX

Built-In Functions Used:

1) **malloc()**
2) **printf()**
3) **scanf()**
4) **exit()**
5) **free()**

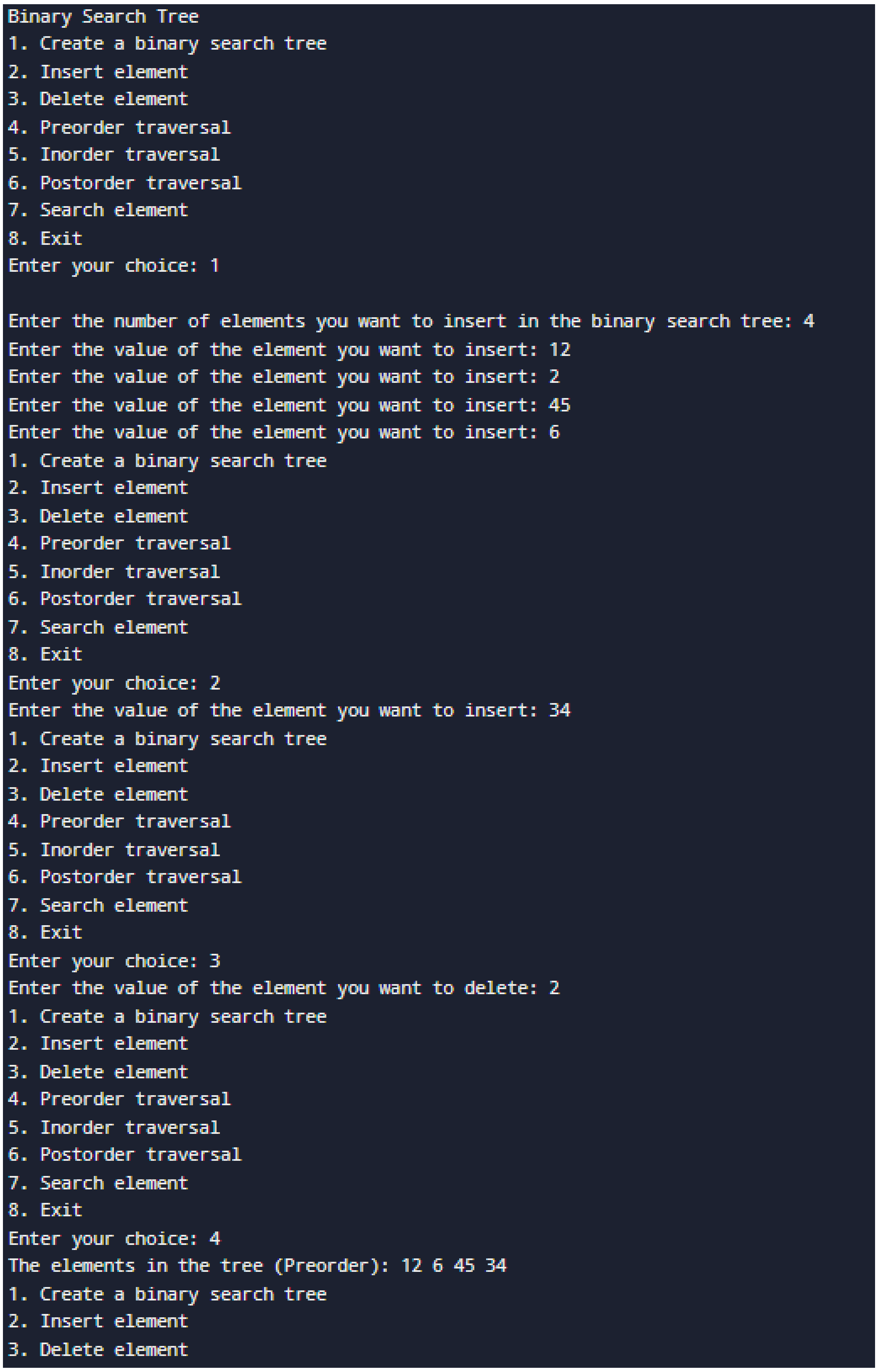
Program source code for Implementation of BST & Binary tree traversal techniques:

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

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

void create_tree(struct node **tree);
struct node *insert_element(struct node *tree, int val);
void preorderTraversal(struct node *tree);
void inorderTraversal(struct node *tree);
void postorderTraversal(struct node *tree);
struct node *del_element(struct node *tree, int val);
struct node *search_element(struct node *tree, int val);

int main() {
    struct node *tree = NULL;
    int choice, val, n;

    printf("Binary Search Tree\n");

    while (1) {
        printf("1. Create a binary search tree\n");
        printf("2. Insert element\n");
        printf("3. Delete element\n");
        printf("4. Preorder traversal\n");
        printf("5. Inorder traversal\n");
        printf("6. Postorder traversal\n");
        printf("7. Search element\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("\nEnter the number of elements you want to insert
in the binary search tree: ");
                scanf("%d", &n);
                for (int i = 0; i < n; i++) {
                    printf("Enter the value of the element you want to
insert: ");
                    scanf("%d", &val);
                    tree = insert_element(tree, val);
                }
                break;

            case 2:
                printf("Enter the value of the element you want to
insert: ");
                scanf("%d", &val);
                tree = insert_element(tree, val);
                break;

            case 3:
                printf("Enter the value of the element you want to
delete: ");
                scanf("%d", &val);
                tree = del_element(tree, val);
                break;

            case 4:
                printf("The elements in the tree (Preorder): ");
                preorderTraversal(tree);
                printf("\n");
                break;

            case 5:
                printf("The elements in the tree (Inorder): ");
```

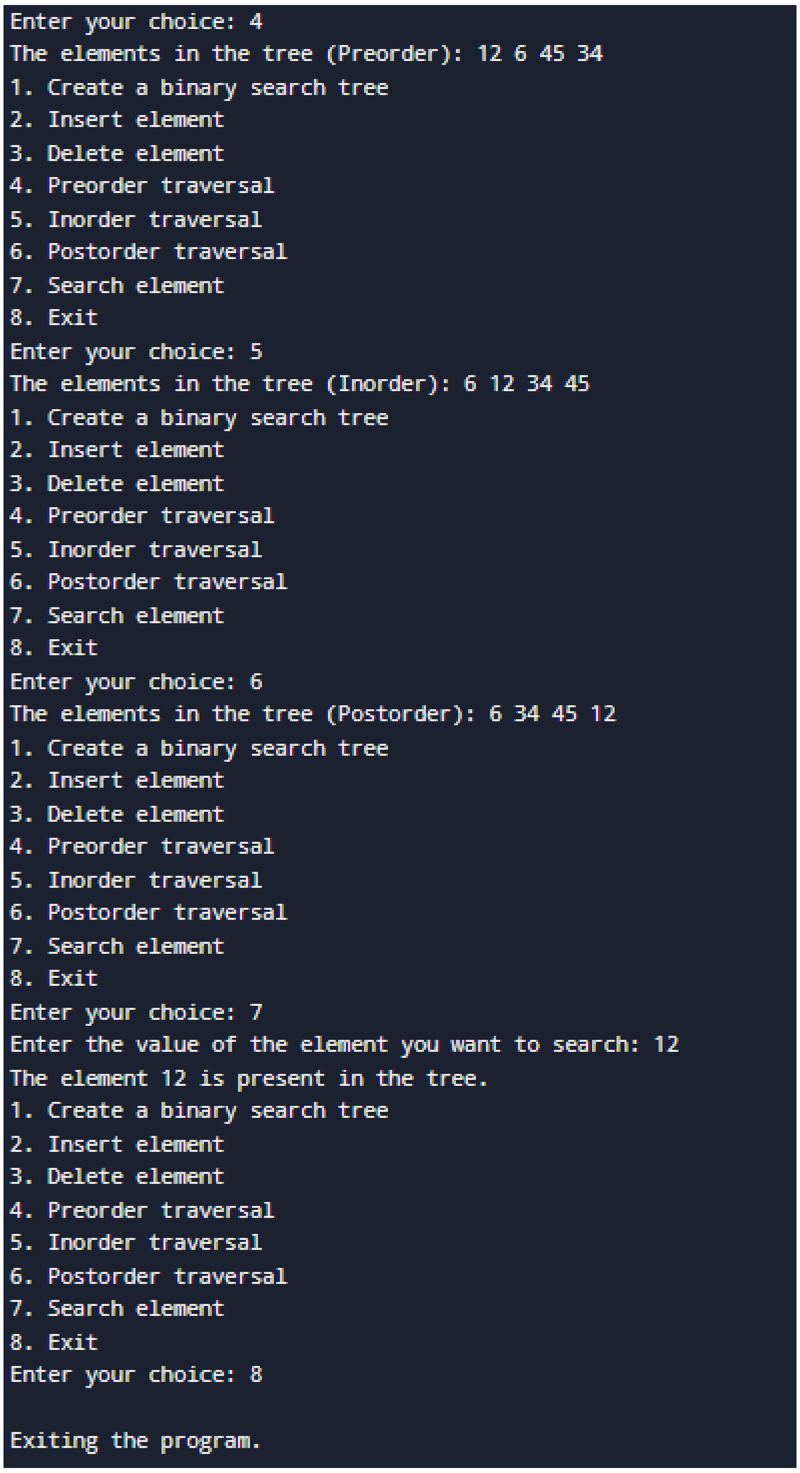
```c
                inorderTraversal(tree);
                printf("\n");
                break;

            case 6:
                printf("The elements in the tree (Postorder): ");
                postorderTraversal(tree);
                printf("\n");
                break;

            case 7:
                printf("Enter the value of the element you want to
search: ");
                scanf("%d", &val);
                if (search_element(tree, val) != NULL) {
                    printf("The element %d is present in the tree.\n",
val);
                } else {
                    printf("The element %d is not present in the
tree.\n", val);
                }
                break;

            case 8:
                printf("\nExiting the program.\n");
                exit(0);
                break;

            default:
                printf("\nInvalid choice! Please try again.\n");
                break;
        }
    }
    return 0;
}

void create_tree(struct node **tree) {
    *tree = NULL;
}

struct node *insert_element(struct node *tree, int val) {
    struct node *ptr = (struct node *)malloc(sizeof(struct node));
    ptr->data = val;
    ptr->left = NULL;
    ptr->right = NULL;
    if (tree == NULL) {
        return ptr;
    } else {
        struct node *parentptr = NULL;
        struct node *nodeptr = tree;

        while (nodeptr != NULL) {
            parentptr = nodeptr;
            if (val < nodeptr->data) {
                nodeptr = nodeptr->left;
            } else if (val > nodeptr->data) {
                nodeptr = nodeptr->right;
            } else {
                printf("The value %d already exists in the tree. Please 
enter a different value.\n", val);
                free(ptr);
                return tree;
            }
        }

        if (val < parentptr->data) {
            parentptr->left = ptr;
        } else {
            parentptr->right = ptr;
        }
    }
    return tree;
}

void preorderTraversal(struct node *tree) {
    if (tree != NULL) {
        printf("%d ", tree->data);
        preorderTraversal(tree->left);
        preorderTraversal(tree->right);
    }
}

void inorderTraversal(struct node *tree) {
    if (tree != NULL) {
        inorderTraversal(tree->left);
        printf("%d ", tree->data);
        inorderTraversal(tree->right);
    }
}

void postorderTraversal(struct node *tree) {
```

```c
    if (tree != NULL) {
        postorderTraversal(tree->left);
        postorderTraversal(tree->right);
        printf("%d ", tree->data);
    }
}

struct node *del_element(struct node *tree, int val) {
    struct node *ptr, *parent, *cur, *suc, *psuc;

    if (tree == NULL) {
        printf("\nThe tree is empty.\n");
        return tree;
    }

    parent = NULL;
    cur = tree;
    while (cur != NULL && val != cur->data) {
        parent = cur;
        cur = (val < cur->data) ? cur->left : cur->right;
    }

    if (cur == NULL) {
        printf("\nThe value %d to be deleted is not present in the
tree.\n", val);
        return tree;
    }

    if (cur->left == NULL) {
        ptr = cur->right;
    } else if (cur->right == NULL) {
        ptr = cur->left;
    } else {
        psuc = cur;
        suc = cur->right;
        while (suc->left != NULL) {
            psuc = suc;
            suc = suc->left;
        }
        if (psuc != cur) {
            psuc->left = suc->right;
        } else {
            psuc->right = suc->right;
        }
        suc->left = cur->left;
        suc->right = cur->right;
        ptr = suc;
    }

    if (parent == NULL) {
        return ptr;
    } else if (parent->left == cur) {
        parent->left = ptr;
    } else {
        parent->right = ptr;
    }

    free(cur);
    return tree;
}

struct node *search_element(struct node *tree, int val) {
    if (tree == NULL) {
        return NULL;
    }

    if (tree->data == val) {
        return tree;
    } else if (val < tree->data) {
        return search_element(tree->left, val);
    } else {
        return search_element(tree->right, val);
    }
}
```

Output Screenshots for Each Operation:



```
Binary Search Tree
1. Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 1

Enter the number of elements you want to insert in the binary search tree: 4
Enter the value of the element you want to insert: 12
Enter the value of the element you want to insert: 2
Enter the value of the element you want to insert: 45
Enter the value of the element you want to insert: 6
1. Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 2
Enter the value of the element you want to insert: 34
1. Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 3
Enter the value of the element you want to delete: 2
1. Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 4
The elements in the tree (Preorder): 12 6 45 34
1. Create a binary search tree
2. Insert element
3. Delete element
```



```
Enter your choice: 4
The elements in the tree (Preorder): 12 6 45 34
1. Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 5
The elements in the tree (Inorder): 6 12 34 45
1. Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 6
The elements in the tree (Postorder): 6 34 45 12
1. Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 7
Enter the value of the element you want to search: 12
The element 12 is present in the tree.
1. Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 8

Exiting the program.
```

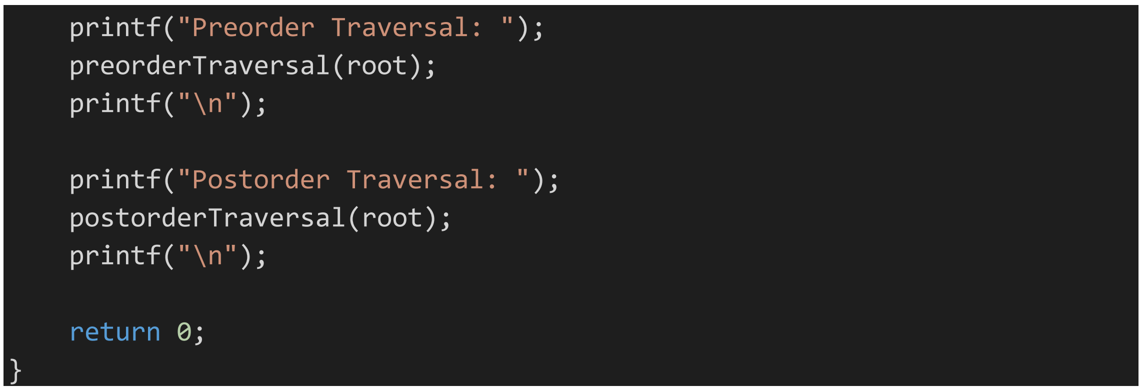
Conclusion:-

Hence, we successfully implemented BST & Binary tree traversal techniques.

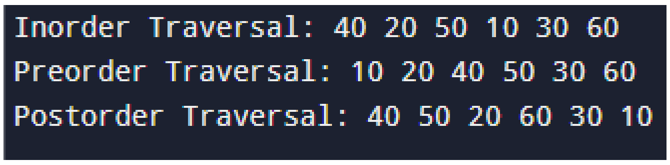
PostLab Questions:

1) **Write a program for inorder, Preorder and Postorder traversal without using recursion.**

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

struct TreeNode {
    int value;
    struct TreeNode* left;
    struct TreeNode* right;
};

struct TreeNode* newNode(int value) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    node->value = value;
    node->left = NULL;
    node->right = NULL;
    return node;
}

void inorderTraversal(struct TreeNode* root) {
    struct TreeNode* stack[100];
    int top = -1;
    struct TreeNode* current = root;

    while (current != NULL || top >= 0) {
        while (current != NULL) {
            stack[++top] = current;
            current = current->left;
        }
        current = stack[top--];
        printf("%d ", current->value);
        current = current->right;
    }
}

void preorderTraversal(struct TreeNode* root) {
    if (root == NULL) return;

    struct TreeNode* stack[100];
    int top = -1;
    stack[++top] = root;

    while (top >= 0) {
        struct TreeNode* current = stack[top--];
        printf("%d ", current->value);

        if (current->right != NULL)
            stack[++top] = current->right;
        if (current->left != NULL)
            stack[++top] = current->left;
    }
}

void postorderTraversal(struct TreeNode* root) {
    if (root == NULL) return;

    struct TreeNode* stack1[100];
    struct TreeNode* stack2[100];
    int top1 = -1, top2 = -1;

    stack1[++top1] = root;

    while (top1 >= 0) {
        struct TreeNode* current = stack1[top1--];
        stack2[++top2] = current;

        if (current->left != NULL)
            stack1[++top1] = current->left;
        if (current->right != NULL)
            stack1[++top1] = current->right;
    }

    while (top2 >= 0) {
        struct TreeNode* current = stack2[top2--];
        printf("%d ", current->value);
    }
}

int main() {
    struct TreeNode* root = newNode(10);
    root->left = newNode(20);
    root->right = newNode(30);
    root->left->left = newNode(40);
    root->left->right = newNode(50);
    root->right->right = newNode(60);

    printf("Inorder Traversal: ");
    inorderTraversal(root);
    printf("\n");
```



```
    printf("Preorder Traversal: ");
    preorderTraversal(root);
    printf("\n");

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

    return 0;
}
```



```
Inorder Traversal: 40 20 50 10 30 60
Preorder Traversal: 10 20 40 50 30 60
Postorder Traversal: 40 50 20 60 30 10
```