ADS LAB ASSINGMENT 3

Name-Sandesh Santosh Kadam
Class- SY IT-C Batch 1
PRN-12210695
Roll no-26

Q. Implement the program in C or C++ : Create BT and insert element of user choice, Non-Recursive
–Inorder, Preorder, Postorder Tree Traversal

Theory-

BST

- A binary search tree is a binary tree in which each node has at most two children, referred to as the left child and the right child.

- For any node in the tree, all the nodes in its left subtree have values less than the node's value, and all the nodes in its right subtree have values greater than the node's value.

- This property allows for efficient search, insertion, and deletion operations.

- Inorder Traversal:

In inorder traversal, the left subtree is visited first, then the current node is processed, and finally, the right subtree is visited. In other words, for a binary tree, the inorder traversal will visit the nodes in ascending order if the values stored in the nodes are numbers.

- Preorder Traversal:

In preorder traversal, the current node is processed first, then the left subtree is visited, and finally, the right subtree is visited. In other words, the preorder traversal visits the nodes in the order of their appearance in the tree.

- Postorder Traversal:

In postorder traversal, the left subtree is visited first, then the right subtree is visited, and finally, the current node is processed. In other words, the postorder traversal visits the nodes in the "bottom-up" manner.

Programming Lang. Used- C

Compiler Used- CodeBlocks

CODE-

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

struct TreeNode {
```

```
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

struct StackNode {
    struct TreeNode* tree_node;
    struct StackNode* next;
};

struct StackNode* createStackNode(struct TreeNode* node) {
    struct StackNode* stackNode = (struct
StackNode*)malloc(sizeof(struct StackNode));
    stackNode->tree_node = node;
    stackNode->next = NULL;
    return stackNode;
}

void push(struct StackNode** top_ref, struct TreeNode*
node) {
    struct StackNode* stackNode = createStackNode(node);
    stackNode->next = *top_ref;
    *top_ref = stackNode;
}

int isEmpty(struct StackNode* top) {
    return top == NULL;
}

struct TreeNode* pop(struct StackNode** top_ref) {
    if (isEmpty(*top_ref))
```

```
        return NULL;

    struct StackNode* temp = *top_ref;
    *top_ref = (*top_ref)->next;
    struct TreeNode* popped = temp->tree_node;
    free(temp);
    return popped;
}
void inorderTraversal(struct TreeNode* root) {
    if (root == NULL)
        return;

    struct TreeNode* current = root;
    struct StackNode* stack = NULL;

    while (current != NULL || !isEmpty(stack)) {
        while (current != NULL) {
            push(&stack, current);
            current = current->left;
        }

        current = pop(&stack);
        printf("%d ", current->data);

        current = current->right;
    }
}

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

```
    struct StackNode* stack = NULL;
    push(&stack, root);

    while (!isEmpty(stack)) {
        struct TreeNode* current = pop(&stack);
        printf("%d ", current->data);

        if (current->right)
            push(&stack, current->right);
        if (current->left)
            push(&stack, current->left);
    }
}

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

    struct StackNode* stack1 = NULL;
    struct StackNode* stack2 = NULL;
    push(&stack1, root);

    while (!isEmpty(stack1)) {
        struct TreeNode* current = pop(&stack1);
        push(&stack2, current);

        if (current->left)
            push(&stack1, current->left);
        if (current->right)
            push(&stack1, current->right);
```

```
    }

    while (!isEmpty(stack2)) {
        struct TreeNode* temp = pop(&stack2);
        printf("%d ", temp->data);
    }
}

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

struct TreeNode* insert(struct TreeNode* 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;
}


int main() {
```

```
    struct TreeNode* root = NULL;
    int i,n,value;

    printf("Enter no of nodes=");
    scanf("%d",&i);
    for(i=1;i<n;i++)
    {
       printf("Enter value of nodes=");
       scanf("%d",&value);
       root = insert(root, value);
    }
    printf("Inorder Traversal: ");
    inorderTraversal(root);
    printf("\n");

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

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


    return 0;
}
```

OUTPUT-


"C:\Users\SANDESH\Documents\psap c\tree traversal using stack.exe"
Enter no of nodes=7
Enter value of nodes=50
Enter value of nodes=30
Enter value of nodes=20
Enter value of nodes=40
Enter value of nodes=70
Enter value of nodes=60
Enter value of nodes=80
Inorder Traversal: 20 30 40 50 60 70 80
Preorder Traversal: 50 30 20 40 70 60 80
Postorder Traversal: 20 40 30 60 80 70 50
Process returned 0 (0x0) execution time : 27.799 s
Press any key to continue.