TREE TRAVERSAL

Aim:

The aim of the provided C code is to implement a binary tree data structure along with functions for creating the tree, and performing pre-order, in-order, and post-order traversals on the tree.

Algorithm:

1. Start
2. Defines a structure Node representing a node in the binary search tree. Each node contains data, left child pointer, and right child pointer.
3. Provides a function to create a new node with the given value and initialize its pointers.
4. To insert a new node into the binary search tree while maintaining the BST property. Create a function to check if the value is less than the current node's data, it traverses to the left subtree; otherwise, it traverses to the right subtree.
5. To delete a node from the binary search tree while preserving the BST property. Create a function to handle cases where the node has zero, one, or two children by finding the successor node and replacing the node to be deleted with it.
6. Create a function to find the node with the minimum value in a subtree, which is used in deletion operation.
7. To search for a value in the binary search tree, Create a recursive function to traverses the tree, comparing the value with each node's data until the value is found or the tree is exhausted.
8. Provide a function to perform an inorder traversal of the binary search tree, printing the nodes in sorted order.
9. End.

Program:

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

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

void create_tree(struct Node*root)
{
    int l,r;
    printf("Enter the value for left child of %d: ",root->data);
    scanf("%d",&l);
    if(l!=0)
```

```
    {
        struct Node*left_node = (struct Node*) malloc (sizeof (struct Node));
        left_node->data=l;
        root->left=left_node;
        create_tree(left_node);
    }
    else root->left=NULL;

    printf("Enter the value for right child of %d: ",root->data);
    scanf("%d",&r);
    if(r!=0)
    {
        struct Node*right_node = (struct Node*) malloc (sizeof (struct Node));
        right_node->data=r;
        root->right=right_node;
        create_tree(right_node);
    }
    else root->right=NULL;
}

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

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

void postorder(struct Node*root)
{
    if(root!=NULL)
    {
```

```
        postorder(root->left);
        postorder(root->right);
        printf("%d ",root->data);
    }
}

void main()
{
    struct Node tree;
    tree.left = NULL;
    tree.right = NULL;
    printf("Enter the data for root node: ");
    scanf("%d",&tree.data);
    create_tree(&tree);
    printf("\nPreorder traversal:\n");
    preorder(&tree);
    printf("\nInorder traversal:\n");
    inorder(&tree);
    printf("\nPostorder traversal:\n");
    postorder(&tree);
    return;
}
```

Output:

```
Enter the data for root node: 1
Enter the value for left child of 1: 2
Enter the value for left child of 2: 4
Enter the value for left child of 4: 8
Enter the value for left child of 8: 0
Enter the value for right child of 8: 0
Enter the value for right child of 4: 0
Enter the value for right child of 2: 5
Enter the value for left child of 5: 0
Enter the value for right child of 5: 10
Enter the value for left child of 10: 0
Enter the value for right child of 10: 0
Enter the value for right child of 1: 3
Enter the value for left child of 3: 6
Enter the value for left child of 6: 0
Enter the value for right child of 6: 0
Enter the value for right child of 3: 7
Enter the value for left child of 7: 0
Enter the value for right child of 7: 9
Enter the value for left child of 9: 0
```

Enter the value for right child of 9: 0

Preorder:
1 2 4 8 5 10 3 6 7 9
Inorder:
8 4 2 5 10 1 6 3 7 9
Postorder:
8 4 10 5 2 6 9 7 3 1
Result:

The output is verified successfully for the above program.

****