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

            // Binary tree Node structure
            struct Node
            {
                int data;
                struct Node *left;
                struct Node *right;
            };

            // ======= Binary Search tree TRAVERSAL ======= //
            // Preorder Traversal
            void preorder(struct Node *root)
            {
                if (root == NULL)
                    return;
                printf("%d ", root->data);
                preorder(root->left);
                preorder(root->right);
            }

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

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

            struct Node *insertNode(struct Node *, int);
            struct Node *createNode(int);

            // ======== Main function ========= //
            int main()
            {
                struct Node *root = NULL;
```

```
    int ch, inputData;
    do
    {
        printf("\n\n1.Insert\n2.Preorder\n3.Inorder\n4.PostOrder\n");
        printf("\nEnter choice  ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("\nEnter data to be inserted: ");
            scanf("%d", &inputData);
            root = insertNode(root, inputData);
            break;
        case 2:
            printf("\nPreorder Traversal\n");
            preorder(root);
            break;
        case 3:
            printf("\nInorder Traversal\n");
            inorder(root);
            break;
        case 4:
            printf("\nPostorder Traversal\n");
            postorder(root);
            break;
        }
    } while (ch < 5);

    return 0;
}

// ====== Additional functions ====== //
// Function to insert a newNode
struct Node *insertNode(struct Node *root, int data)
{
    if (root == NULL)
        root = createNode(data);
    else if (data <= root->data)
        root->left = insertNode(root->left, data);
    else
        root->right = insertNode(root->right, data);

    return root;
}
// 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;
```

```
}
```

Output

```
1.Insert
2.Preorder
3.Inorder
4.PostOrder
Enter choice  1
Enter data to be inserted: 1
1.Insert
2.Preorder
3.Inorder
4.PostOrder

Enter choice  1
Enter data to be inserted: 2
1.Insert
2.Preorder
3.Inorder
4.PostOrder

Enter choice  1
Enter data to be inserted: 3
1.Insert
2.Preorder
```

3.Inorder

4.PostOrder

Enter choice 1

Enter data to be inserted: 7

1.Insert

2.Preorder

3.Inorder

4.PostOrder

Enter choice 1

Enter data to be inserted: 9

1.Insert

2.Preorder

3.Inorder

4.PostOrder

Enter choice 1

Enter data to be inserted: 5

1.Insert

2.Preorder

3.Inorder

4.PostOrder

Enter choice 1

Enter data to be inserted: 8

1.Insert

2.Preorder

3.Inorder

4.PostOrder

Enter choice 2

Preorder Traversal

1 2 3 7 5 9 8

1.Insert

2.Preorder

3.Inorder

4.PostOrder

Enter choice 2

Preorder Traversal

1 2 3 7 5 9 8

1.Insert

2.Preorder

3.Inorder

4.PostOrder

Enter choice 3

Inorder Traversal

1 2 3 5 7 8 9

1.Insert

2.Preorder

3.Inorder

4.PostOrder

Enter choice 4

Postorder Traversal

5 8 9 7 3 2 1

1.Insert

2.Preorder

3.Inorder

4.PostOrder

Enter choice