```c
// Write a program in C to implement a Binary Search Tree (BST) using Linked
List
// and carry out the following operations in it.
// a)Construct a BST for some given elements
// b)Insert a node into the BST
// c)Delete a node from the BST
// d)Traverse the BST in Inorder
// e)Traverse the BST in Preorder
// f)Traverse the BST in Postorder
// g)Find Minimum element in the BST
// h)Find Maximum element in the BST

#include <stdio.h>

#include <stdlib.h>

struct BST

{
    int data;

    struct BST *left;

    struct BST *right;
};
typedef struct BST NODE;

NODE *node;

NODE *createtree(NODE *node, int data);
NODE *search(NODE *node, int data);
NODE *insert(NODE *node, int);
void inorder(NODE *node);
void preorder(NODE *node);
void postorder(NODE *node);
NODE *findMin(NODE *node);
NODE *findMax(NODE *node);
NODE *del(NODE *node, int data);

void main()

{

    int data, ch, i, n, min, max;

    NODE *root = NULL;

    while (1)
```

```
    {

        printf("\n1. Create BST");

        printf("\n2. Insert a node into BST");

        printf("\n3. Delete a node from the BST");

        printf("\n4. Search an element");

        printf("\n5. Traverse the BST in Inorder\n6. Traverse the BST in
Preorder");

        printf("\n7. Traverse the BST in Postorder\n8. Find Minimum element in
the BST\n9. Find Maximum element in the BST");

        printf("\n8. Exit\n");

        printf("\nEnter your choice: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:
            printf("\nEnter N value: ");

            scanf("%d", &n);

            printf("\nEnter the values to create BST
like(6,9,5,2,8,15,24,14,7,8,5,2)\n");

            for (i = 0; i < n; i++)

            {

                scanf("%d", &data);

                root = createtree(root, data);
            }

            break;

        case 2:
```

```
            printf("Enter the value to be inserted\n");

            scanf("%d",&data);

            root = insert(root, data);

            break;

        case 3:

            printf("\nEnter the element to delete: ");

            scanf("%d", &data);

            root = del(root, data);

            break;

        case 4:
            printf("\nEnter the element to search: ");

            scanf("%d", &data);

            root = search(root, data);

            break;


        case 5:
            printf("\nInorder Traversal: \n");

            inorder(root);

            break;

        case 6:
            printf("\nPreorder Traversal: \n");

            preorder(root);

            break;

        case 7:
            printf("\nPostorder Traversal: \n");

            postorder(root);

            break;
```

```
        case 8:
            root = findMin(root);

            printf("The minimum element in the BST is: %d",root);

            break;

        case 9:
            root = findMax(root);

            printf("The minimum element in the BST is: %d",root);

            break;

        case 10:
            exit(0);

        default:
            printf("\nWrong option");

            break;
        }
    }
}
NODE *createtree(NODE *node, int data)

{

    if (node == NULL)

    {

        NODE *temp;

        temp = (NODE *)malloc(sizeof(NODE));

        temp->data = data;

        temp->left = temp->right = NULL;

        return temp;
    }

    if (data < (node->data))

    {
```

```
        node->left = createtree(node->left, data);
    }

    else if (data > node->data)

    {

        node->right = createtree(node->right, data);
    }

    return node;
}

NODE *search(NODE *node, int data)

{

    if (node == NULL)

        printf("\nElement not found");

    else if (data < node->data)

    {

        node->left = search(node->left, data);
    }

    else if (data > node->data)

    {

        node->right = search(node->right, data);
    }

    else

        printf("\nElement found is: %d", node->data);

    return node;
}
NODE *insert(NODE *node, int data)
{

    // if (node == NULL)
    //     return getNewNode(val);

    if (node->data < data)
```

```
        node->right = insert(node->right, data);

    else if (node->data > data)
        node->left = insert(node->left, data);

    return node;
}

void inorder(NODE *node)
{

    if (node != NULL)
    {
        inorder(node->left);

        printf("%d\t", node->data);

        inorder(node->right);
    }
}

void preorder(NODE *node)

{

    if (node != NULL)

    {

        printf("%d\t", node->data);

        preorder(node->left);

        preorder(node->right);
    }
}

void postorder(NODE *node)

{

    if (node != NULL)

    {

        postorder(node->left);

        postorder(node->right);
```

```
        printf("%d\t", node->data);
    }
}

NODE *findMin(NODE *node)

{

    if (node == NULL)

    {

        return NULL;
    }

    if (node->left)

        return findMin(node->left);

    else

        return node;
}
NODE *findMax(NODE *node)
{
    if (node == NULL)

        return NULL;

    if (node->right)

        return findMin(node->right);

    else

        return node;
}

NODE *del(NODE *node, int data)

{

    NODE *temp;

    if (node == NULL)

    {
```

```
        printf("\nElement not found");
    }

    else if (data < node->data)

    {

        node->left = del(node->left, data);
    }

    else if (data > node->data)

    {

        node->right = del(node->right, data);
    }

    else

    {
        if (node->right && node->left)

        {
            temp = findMin(node->right);

            node->data = temp->data;

            node->right = del(node->right, temp->data);
        }

        else

        {

            temp = node;

            if (node->left == NULL)

                node = node->right;

            else if (node->right == NULL)

                node = node->left;

            free(temp);
        }
    }
```

```
    return node;
}
```