Name: Shreeharsh
USN: 1BM19CS153

Binary Search Tree

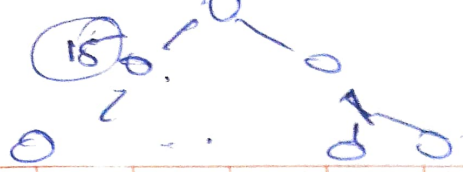
```
#include<stdlib.h>
#include<math.h>

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

NODE getnode (int data)
{
    NODE x = (NODE) malloc(sizeof
                (struct node));

    x -> data = data;
    x -> right = NULL;
    x -> left = NULL;

}
```

```
NODE insert (NODE root, int info)
{
    NODE new_node = getnode(info)
    if (new
    {
    }
    if (root == NULL)
        NODE new_node = getnode(info);
    else if (root -> data <= info)
        root -> right = insert(root
                               -> right, info
    else
        root -> left = insert(root -> left, inf
);
}
```

```
Void preorder(NODE root)
{ if (root != NULL) {
    printf ("%d\t", root->data);
    preorder (root -> left);
    preorder (root -> right);
  }
}

Void postorder(NODE root)
{
    if (root == NULL)
        return

    postorder (root -> left);
    postorder (root -> right);
}
```

```
void inorder (NODE root)
{
    if (root == NULL)
        return

    inorder (root -> left);
    printf ("%d ", root -> data);
    inorder (root -> right);

}
NODE delete_newnode (NODE root, data);
{
    if (root == NULL)
        return root
else if (info < root -> data)
        root -> left = delete_newnode
                        (root -> left data

else if (info > root -> data
        root -> right = delete-newnode
                        (root -> right,
                        data)

else
```

```
if (root -> right == NULL && root -> left == NULL)
{
    NODE temp = root;
    free(temp);
    root = NULL;
    return true;
}
else if (root -> right == NULL)
{
    NODE temp = root;
    free(temp);
    root = root -> left;
    return root;
}
else if (left -> right == NULL)
{
    NODE temp = root;
    free(temp);
    root = root -> right;
    return root
}
```



Page No.
Date / /

root -> left == NULL)

else
{

NODE temp = findmin(root->right)

root -> data = temp -> data;
root -> right = delete
(root->right-
,temp->d

}
}return root

{

```
void display(NODE root, int i)
{
    if (root == NULL)
        return

    display(root->right, i+1);
    for(int j=1; j<=i; j++)
        printf(" ");
    printf("%d\n", root->data);
    display(root->left, i+1);
    display(root->left, i+1);

}

int height(NODE root)
{
    if (root == NULL)
        return -1;
    int left_height = height(root->left);
    int right_height = height(root->right);
    int val = max(left_height, right_height);
    return val + 1;
}
```

get

```
int leafCount(NODE root)
{
    if (root == NULL)
        return 0;
    if ( root->right == NULL &&
         root->left
         == NULL)

        return 1

    else
        return (getCount( root->right)
                + getCount(root->left));

}
```

```
int main()
{
    printf("1: insert
do {
    NODE root = NULL;
    int option, data;
    printf("1: insert \n");
    printf("2: Delete \n");
    printf("3: Preorder\n");
    printf("4: Postorder\n");
    printf("5: inorder \n");
    printf("6: display\n");
    printf("Enter your option");
    scanf("%d", &option);
    switch(option)
    {
        case 1:
            printf("Enter data\n");
            scanf("%d", &data);
            root = insert(root, data);
            break;

        case 2:
            printf("enter data\n");
            scanf("%d", &data);
```

```
            root = delete_node(root, data);
            break;

        case 3:
            preorder(root);
            break;

        case 4:
            postorder(root);
            break;

        case 5:
            inorder(root);
            break;

        case 6:
            display(root);
            break;
        }
);  } while(option != 7);
    }
{
```