Name – Shrinivas Hatyalikar
Div – CS-B
Roll no – 24

Perform following operations on BST.
a. Create
b. Insert
c. Delete
d. Mirror Image
e. Level wise Display
f. Height of the tree
g. Display Leaf Nodes

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

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

struct node *create(int data)
{
   struct node *new_node = (struct node *)malloc(sizeof(struct node));
   new_node->data = data;
   new_node->left = NULL;
   new_node->right = NULL;
   return new_node;
}

struct node *insert(struct node *root, int data)
{
   if (root == NULL)
   {
      return create(data);
   }
```

```
    if (data < root->data)
    {
        root->left = insert(root->left, data);
    }
    else if (data > root->data)
    {
    root->right = insert(root->right, data);
    }
    return root;
}

struct node *delete(struct node *root, int data)
{
    if (root == NULL)
    {
        return root;
    }
    if (data < root->data)
    {
        root->left = delete (root->left, data);
    }
    else if (data > root->data)
    {
        root->right = delete (root->right, data);
    }
    else
    {
    if (root->left == NULL)
    {
        struct node *temp = root->right;
        free(root);
        return temp;
    }
    else if (root->right == NULL){
        struct node *temp = root->left;
        free(root);
            return temp;
     }
     struct node *temp = root->right;
     while (temp && temp->left != NULL)
```

```
    {
    temp = temp->left;
   }
   root->data = temp->data;
   root->right = delete (root->right, temp->data);
    }
    return root;
}

void mirror(struct node *root)
{
   if (root == NULL)
    {
    return;
    }
   mirror(root->left);
    mirror(root->right);
    struct node *temp = root->left;
    root->left = root->right;
    root->right = temp;
   }

int height(struct node *root)
{
    if (root == NULL)
 {
 return 0;
}
 int left_height = height(root->left);
 int right_height = height(root->right);
 if (left_height > right_height)
 {
 return left_height + 1;
 }
 else
 {
 return right_height + 1;
 }
}
```

```
void display_leaf_nodes(struct node *root)
{
 if (root == NULL)
 {
 return;
 }
 if (root->left == NULL && root->right == NULL)
 {
 printf("%d ", root->data);
 }
 display_leaf_nodes(root->left);
 display_leaf_nodes(root->right);
}

void level_order_traversal(struct node *root)
{
 if (root == NULL)
 {
 return;
 }
 struct node *queue[100];
 int front = -1;
 int rear = -1;
 queue[++rear] = root;
 while (front != rear)
 {
 struct node *current = queue[++front];
 printf("%d ", current->data);
 if (current->left != NULL)
 {
 queue[++rear] = current->left;
 }
if (current->right != NULL)
 {
 queue[++rear] = current->right;
 }
}
}

int main()
```

```
{
 struct node *root = NULL;
 root = insert(root, 100);
 insert(root, 200);
 insert(root, 300);
 insert(root, 400);
 insert(root, 500);
 insert(root, 600);
 insert(root, 700);

 printf("Level order traversal: ");
 level_order_traversal(root);
 printf("\n");

 printf("Height of the tree: %d\n", height(root));
 printf("Leaf nodes: ");
 display_leaf_nodes(root);
 printf("\n");

 printf("Deleting node 30: \n");
 root = delete (root, 30);
printf("done!\n");
 printf("Level order traversal after deleting node 30: ");
 level_order_traversal(root);
printf("\n");

 printf("Mirroring the tree-> \n");
mirror(root);
 printf("done!\n");

 printf("Level order traversal after mirroring: ");
 level_order_traversal(root);
 printf("\n");

 return 0;
}
```

```
PS C:\Users\sheeh\OneDrive\Desktop\C> cd "c:\Users\sheeh\OneDrive\Desktop\C"
PS C:\Users\sheeh\OneDrive\Desktop\C> cd "c:\Users\sheeh\OneDrive\Desktop\C\" ; if ($?) { gcc BST.c -o BST } ; if ($?) { .\BST }
Level order traversal: 100 200 300 400 500 600 700
Height of the tree: 7
Leaf nodes: 700
Deleting node 30:
done!
Level order traversal after deleting node 30: 100 200 300 400 500 600 700
Mirroring the tree->
done!
Level order traversal after mirroring: 100 200 300 400 500 600 700
PS C:\Users\sheeh\OneDrive\Desktop\C>
```