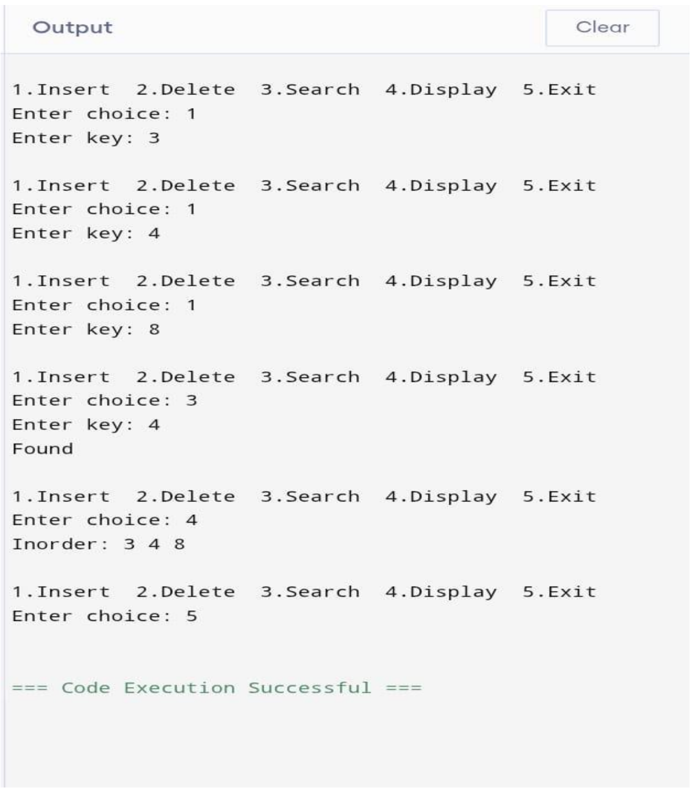
CSA 0317 DATA STRUCTURES

PROGRAM 20

```
#include <stdio.h>

#include <stdlib.h>


struct node {

   int key, height;

   struct node *left, *right;

};


int height(struct node *n) {

   return n ? n->height : 0;

}


int max(int a, int b) { return (a > b) ? a : b; }


struct node *newNode(int key) {

   struct node temp = (struct node)malloc(sizeof(struct node));

   temp->key = key;

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

   temp->height = 1;

   return temp;

}


struct node *rightRotate(struct node *y) {

   struct node *x = y->left;

   y->left = x->right;

   x->right = y;

   y->height = max(height(y->left), height(y->right)) + 1;

   x->height = max(height(x->left), height(x->right)) + 1;
```

```
    return x;
}

struct node *leftRotate(struct node *x) {
    struct node *y = x->right;
    x->right = y->left;
    y->left = x;
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
    return y;
}

int getBalance(struct node *n) {
    return n ? height(n->left) - height(n->right) : 0;
}

struct node *insert(struct node *node, int key) {
    if (!node) return newNode(key);
    if (key < node->key) node->left = insert(node->left, key);
    else if (key > node->key) node->right = insert(node->right, key);
    else return node;

    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);

    if (balance > 1 && key < node->left->key) return rightRotate(node);
    if (balance < -1 && key > node->right->key) return leftRotate(node);
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}

struct node *minValueNode(struct node *n) {
    while (n->left) n = n->left;
    return n;
}

struct node *deleteNode(struct node *root, int key) {
    if (!root) return root;
    if (key < root->key) root->left = deleteNode(root->left, key);
    else if (key > root->key) root->right = deleteNode(root->right, key);
    else {
        if (!root->left || !root->right) {
            struct node *temp = root->left ? root->left : root->right;
            if (!temp) { temp = root; root = NULL; }
            else *root = *temp;
            free(temp);
        } else {
            struct node *temp = minValueNode(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
        }
    }
    if (!root) return root;

    root->height = 1 + max(height(root->left), height(root->right));
    int balance = getBalance(root);

    if (balance > 1 && getBalance(root->left) >= 0) return rightRotate(root);
    if (balance > 1 && getBalance(root->left) < 0) {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    if (balance < -1 && getBalance(root->right) <= 0) return leftRotate(root);
    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}

int search(struct node *root, int key) {
    if (!root) return 0;
    if (root->key == key) return 1;
    if (key < root->key) return search(root->left, key);
    return search(root->right, key);
}

void inorder(struct node *root) {
    if (root) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}

int main() {
    struct node *root = NULL;
    int ch, key;


    while (1) {
        printf("\n1.Insert  2.Delete  3.Search  4.Display  5.Exit\nEnter choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1: printf("Enter key: "); scanf("%d", &key); root = insert(root, key); break;
            case 2: printf("Enter key: "); scanf("%d", &key); root = deleteNode(root, key); break;
            case 3: printf("Enter key: "); scanf("%d", &key);
                    if (search(root, key)) printf("Found\n"); else printf("Not Found\n"); break;
            case 4: printf("Inorder: "); inorder(root); printf("\n"); break;
            case 5: exit(0);
        }
    }
}
```

Output:



```
1.Insert  2.Delete  3.Search  4.Display  5.Exit
Enter choice: 1
Enter key: 3

1.Insert  2.Delete  3.Search  4.Display  5.Exit
Enter choice: 1
Enter key: 4

1.Insert  2.Delete  3.Search  4.Display  5.Exit
Enter choice: 1
Enter key: 8

1.Insert  2.Delete  3.Search  4.Display  5.Exit
Enter choice: 3
Enter key: 4
Found

1.Insert  2.Delete  3.Search  4.Display  5.Exit
Enter choice: 4
Inorder: 3 4 8

1.Insert  2.Delete  3.Search  4.Display  5.Exit
Enter choice: 5


=== Code Execution Successful ===
```