Name	Komal Tarachandani
UID no.	2021600065
Experiment No.	6

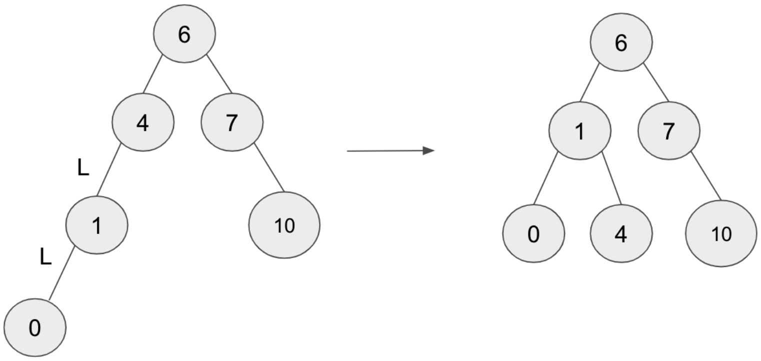
AIM:	To implement creation and deletion in AVL trees
Program6	
PROBLEM STATEMENT :	1. Create 2. Insert (LL , LR , RL , RR rule) 3. Display 1. Deletion with all cases

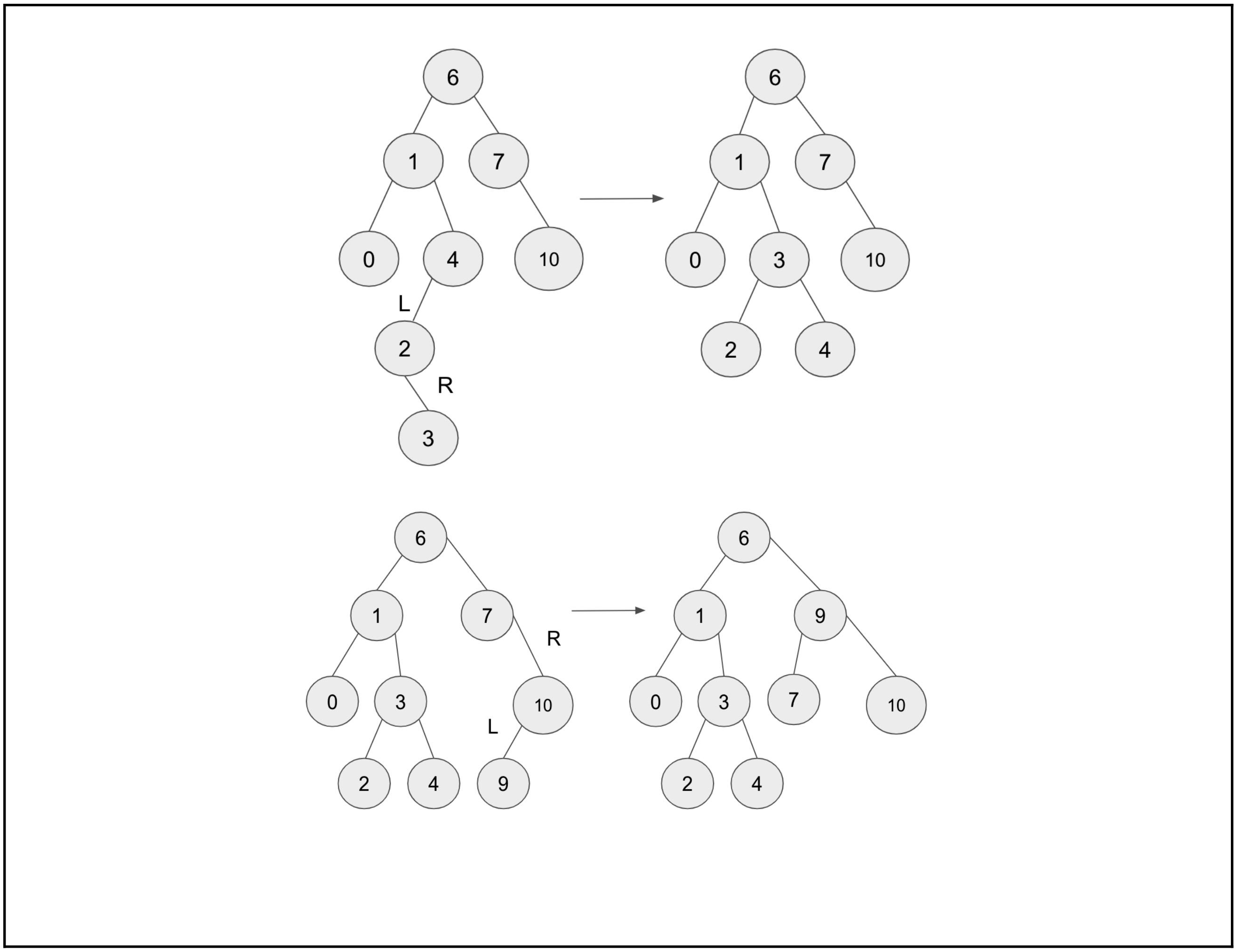
PROGRAM:

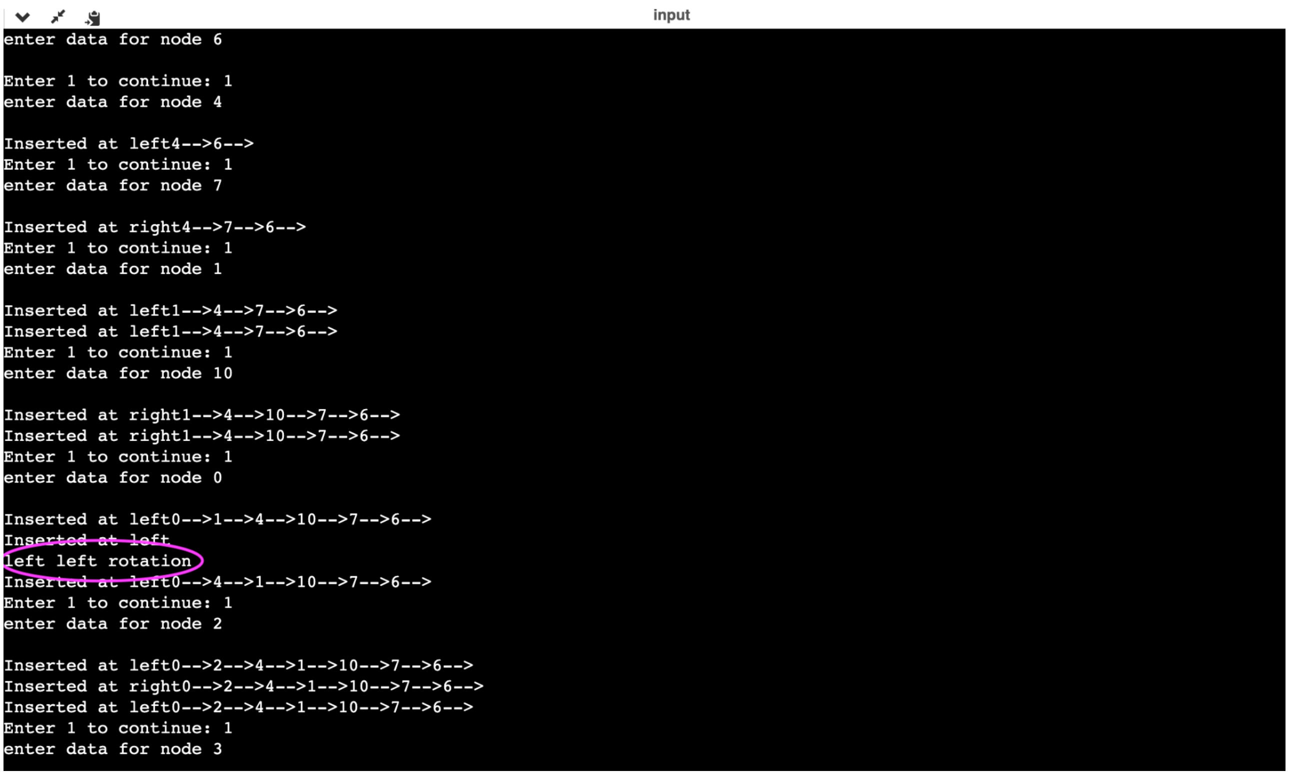
```
#include <iostream>
using namespace std;

// An AVL tree node
class Node
{
    public:
    int data;
    Node *left;
    Node *right;
    Node *parent;
    int height;
};

int height(Node *N)
{
    if (N == NULL)
        return 0;
    return N->height;
}

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

Node* newNode(int key)
{
    Node* node = new Node();
    node->data = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return(node);
}

Node *rightRotate(Node *y)
{
    Node *x = y->left;
    Node *T2 = x->right;

    x->right = y;
    y->left = T2;

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


    return x;
}


Node *leftRotate(Node *x)
{
    Node *y = x->right;
    Node *T2 = y->left;


    y->left = x;
    x->right = T2;


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


    // Return new root
    return y;
}


// Get Balance factor of node N
int getBalance(Node *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}
```

```
Node* insert(Node* node, int num)
{
    if (node == NULL)
        return(newNode(num));

    if (num < node->data)
        {
            node->left = insert(node->left, num);
            cout<<"Inserted at left";
        }
    else if (num > node->data)
    {
        node->right = insert(node->right, num);
        cout<<"Inserted at right";
    }
    else
        return node;

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

    // Left Left Case
    if (balance > 1 && num < node->left->data)
    {
        cout<<"\nleft left rotation";
        return rightRotate(node);
    }
    // Right Right Case
    if (balance < -1 && num > node->right->data)
    {
        cout<<"\nright right rotation";
        return leftRotate(node);
    }
    // Left Right Case
    if (balance > 1 && num > node->left->data)
    {
        cout<<"\nleft right rotation";
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    // Right Left Case
    if (balance < -1 && num < node->right->data)
    {
        cout<<"\nleft left rotation";
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}
void display(Node *cur)
{
    if(cur==NULL)
    {
        return ;
    }
    display(cur->left);
    display(cur->right);
    cout<<cur->data<<"-->";
}
Node *minValueNode(Node *node)
{
    Node* current = node;

    /* loop down to find the leftmost leaf */
    while (current && current->left != NULL)
        current = current->left;

    return current;
}
Node* deleteNode(Node *root, int key)
{
    // base case
    if (root == NULL)
        return root;



    if (key < root->data)
        root->left = deleteNode(root->left, key);

    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {

        if (root->left==NULL and root->right==NULL)
            return NULL;


        else if (root->left == NULL) {
            Node* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            Node* temp = root->left;
            free(root);
            return temp;
        }


        Node* temp = minValueNode(root->right);

        root->data = temp->data;

        // Delete the inorder successor
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
int main()
{
    Node *root;
    int ch=1,num;
    root=NULL;
    while(ch==1)
    {
        cout<<"enter data for node ";
        cin>>num;
        root=insert(root,num);
        cout<<"\nEnter 1 to continue: ";
        cin>>ch;
    }
    display(root);


    root=deleteNode(root,6);
    display(root);
    return 0;
}
```

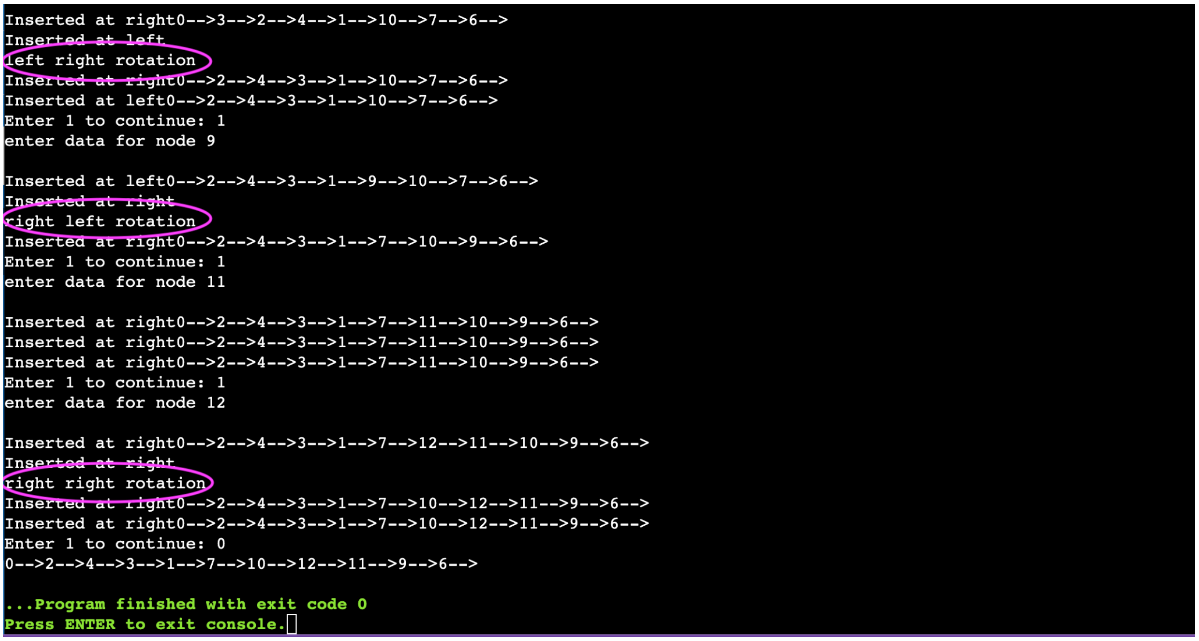
RESULT: TREE:



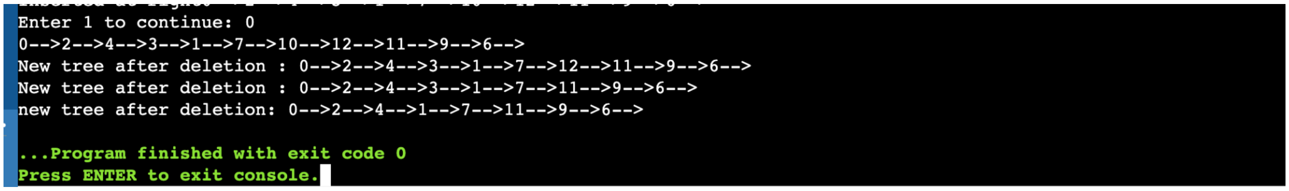

6
1
7
0
4
10
L
2
R
3
6
1
7
0
3
10
2
4
6
1
7
R
0
3
10
L
2
4
9
6
1
9
0
3
7
10
2
4



```
enter data for node 6

Enter 1 to continue: 1
enter data for node 4

Inserted at left4-->6-->
Enter 1 to continue: 1
enter data for node 7

Inserted at right4-->7-->6-->
Enter 1 to continue: 1
enter data for node 1

Inserted at left1-->4-->7-->6-->
Inserted at left1-->4-->7-->6-->
Enter 1 to continue: 1
enter data for node 10

Inserted at right1-->4-->10-->7-->6-->
Inserted at right1-->4-->10-->7-->6-->
Enter 1 to continue: 1
enter data for node 0

Inserted at left0-->1-->4-->10-->7-->6-->
Inserted at left
left left rotation
Inserted at left0-->4-->1-->10-->7-->6-->
Enter 1 to continue: 1
enter data for node 2

Inserted at left0-->2-->4-->1-->10-->7-->6-->
Inserted at right0-->2-->4-->1-->10-->7-->6-->
Inserted at left0-->2-->4-->1-->10-->7-->6-->
Enter 1 to continue: 1
enter data for node 3
```



```
Inserted at right0-->3-->2-->4-->1-->10-->7-->6-->
Inserted at left
left right rotation
Inserted at right0-->2-->4-->3-->1-->10-->7-->6-->
Inserted at left0-->2-->4-->3-->1-->10-->7-->6-->
Enter 1 to continue: 1
enter data for node 9

Inserted at left0-->2-->4-->3-->1-->9-->10-->7-->6-->
Inserted at right
right left rotation
Inserted at right0-->2-->4-->3-->1-->7-->10-->9-->6-->
Enter 1 to continue: 1
enter data for node 11

Inserted at right0-->2-->4-->3-->1-->7-->11-->10-->9-->6-->
Inserted at right0-->2-->4-->3-->1-->7-->11-->10-->9-->6-->
Inserted at right0-->2-->4-->3-->1-->7-->11-->10-->9-->6-->
Enter 1 to continue: 1
enter data for node 12

Inserted at right0-->2-->4-->3-->1-->7-->12-->11-->10-->9-->6-->
Inserted at right
right right rotation
Inserted at right0-->2-->4-->3-->1-->7-->10-->12-->11-->9-->6-->
Inserted at right0-->2-->4-->3-->1-->7-->10-->12-->11-->9-->6-->
Enter 1 to continue: 0
0-->2-->4-->3-->1-->7-->10-->12-->11-->9-->6-->

...Program finished with exit code 0
Press ENTER to exit console.
```



```
Enter 1 to continue: 0
0-->2-->4-->3-->1-->7-->10-->12-->11-->9-->6-->
New tree after deletion : 0-->2-->4-->3-->1-->7-->12-->11-->9-->6-->
New tree after deletion : 0-->2-->4-->3-->1-->7-->11-->9-->6-->
new tree after deletion: 0-->2-->4-->1-->7-->11-->9-->6-->

...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION:	Hence I was able to learn the proper implementation and application of AVL TREES.