```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>

//create a structre of node
class node
{
        public:
                int data;
                int bal;
                node *left;      //left link
                node *right;    //right link

};

class Tree
{
public:
        node *insert(node *,node *); //function to insert node
        node *rotateRight(node *);
        node *rotateLeft(node *);
        void disp(node *,int);  //function to display tree

};

//function to display tree
void Tree::disp(node* root,int k)
{
        int i;
        if(root)
        {
                disp(root->right, k+1);
                cout<<endl;
                for(i = 0; i< k; i++)
                        cout<<' ';
                cout<<root->data;
                disp(root->left, k+1);
        }
}

//function to insert the node into tree
node *Tree::insert(node *root,node *s)
{
        //insert at right
        if(s->data > root->data)
        {
                if(root->right == NULL)
                        root->right = s;
```

```
        else
                root->right = insert(root->right, s);
}

//insert at left
if(s->data <= root->data)
{
        if(root->left == NULL)
                root->left = s;
        else
                root->left = insert(root->left,s);
}

//update the balance factor from leaf node
if(root->left==NULL && root->right!=NULL)
        root->bal = -1;
else if(root->left!=NULL && root->right==NULL)
        root->bal = 1;
else
        root->bal = 0;

//rebalance the tree if node is unbalanced
//Case 1: Left of Left
if(root->bal == 1 && root->left->bal == 1)
    root = rotateRight(root);

//Case 2: Right of Right
if(root->bal == -1 && root->right->bal == -1)
        root = rotateLeft(root);

//Case 3: Right of Left
if(root->bal == 1 && root->left->bal == -1)
{
        root->left = rotateRight(root->left);
        root = rotateRight(root);
}

//Case 4: Left of Right
if(root->bal == -1 && root->right->bal == 1)
{
                root->right = rotateLeft(root->right);
                root = rotateRight(root);
        }
        return root;

}

node *Tree :: rotateRight(node *root)
{
```

```
        node *temp;
        temp = root->left;
        root->left = temp->right;
        temp->right = root;
        return temp;
}

node *Tree :: rotateLeft(node *root)
{
        node *temp;
        temp = root->right;
        root->right = temp->left;
        temp->left = root;
        return temp;
}

void main()
{
        int ch,c;
        char temp;
        node *root,*s;
        Tree t;

        root = new node;       //allocate memory for new node
        root = NULL;   //initially root is NULL

        clrscr();

        do{
                s=new node;  //allocate memory for new node
                s->bal = 0;
                s->left=NULL; //assign left child as NULL
                s->right=NULL;          //assign right child as NULL

                cout<<"\n"<<"Enter node of tree::";
                cin>>s->data;

                //check whether root is NULL
                if(root == NULL)
                        root = s;
                else
                        root = t.insert(root, s);

                cout<<"\n\n Tree is :\n\n";
                t.disp(root, 1);
                cout<<"\n";
                cout<<"\n"<<"WANT TO ENTER MORE ELEMENTS(y/n)::";
                cin>>temp;
```

```
        }while(temp=='y');

        //t.disp(root,1);

        cout<<endl;
}
```