```c
//Author:DEADPOOL
//User@DEADPOOL
//Device name:LAPTOP-MGJPSU5N
//**************************
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<time.h>
#include<windows.h>
typedef struct tnode{
    int value,height;
    struct tnode *left;
    struct tnode *right;
}tnode;
tnode* create_tnode(int value){
    tnode* new_node =malloc(sizeof(tnode));
    if (new_node!=NULL){
        new_node->left=NULL;
        new_node->right=NULL;
        new_node->value=value;
        new_node->height=1;
    }
    return new_node;
}
int height(tnode *N)
{
    if (N == NULL)
        return 0;
    return N->height;
}


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

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

    // Perform rotation
    x->right = y;
    y->left = T2;

    // Update heights
    y->height = int_max(height(y->left), height(y->right))+1;
    x->height = int_max(height(x->left), height(x->right))+1;

    // Return new root
    return x;
}

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

    // Perform rotation
    y->left = x;
    x->right = T2;

    //  Update heights
    x->height = int_max(height(x->left), height(x->right))+1;
    y->height = int_max(height(y->left), height(y->right))+1;
```

```
67
68      // Return new root
69      return y;
70  }
71
72  int getBalance(tnode *N)
73  {
74      if (N == NULL)
75          return 0;
76      return height(N->left) - height(N->right);
77  }
78
79  tnode* insert(tnode* node, int value)
80  {
81      /* 1.  Perform the normal BST insertion */
82      if (node == NULL)
83          return(create_tnode(value));
84
85      if (value < node->value)
86          node->left  = insert(node->left, value);
87      else if (value > node->value)
88          node->right = insert(node->right, value);
89      else // Equal values are not allowed in BST
90          return node;
91
92      /* 2. Update height of this ancestor node */
93      node->height = 1 + int_max(height(node->left),
94                                 height(node->right));
95
96      /* 3. Get the balance factor of this ancestor
97            node to check whether this node became
98            unbalanced */
99      int balance = getBalance(node);
100
101     // If this node becomes unbalanced, then
102     // there are 4 cases
103
104     // Left Left Case
105     if (balance > 1 && value < node->left->value)
106         return rightRotate(node);
107
108     // Right Right Case
109     if (balance < -1 && value > node->right->value)
110         return leftRotate(node);
111
112     // Left Right Case
113     if (balance > 1 && value > node->left->value)
114     {
115         node->left =  leftRotate(node->left);
116         return rightRotate(node);
117     }
118
119     // Right Left Case
120     if (balance < -1 && value < node->right->value)
121     {
122         node->right = rightRotate(node->right);
123         return leftRotate(node);
124     }
125
126     /* return the (unchanged) node pointer */
127     return node;
128 }
129
130 //functions to print a tree (AVL tree)
131 //this delay function will helps to view the output data properly
132 void delay(unsigned int mseconds)
```

```
{
    clock_t goal = mseconds + clock();
    while (goal > clock());
}

void print_format(int num_of_char){
        delay(45);
        printf("%c",219);
    for(int i=0;i<num_of_char;i++){
        delay(90);
        printf("        %c",219);
    }
    printf("%c%c%c%c",254,254,254,254);
}
// pre-order traversal
void pre_order_print_tree(tnode* root,int level){
    if (root==NULL){
        print_format(level);
        printf("...\n");
        return;
    }
    print_format(level);
    printf("%d(L:%d)\n",root->value,level);
    print_format(level);
    printf(" Left\n");
    pre_order_print_tree(root->left,level+1);
    print_format(level);
    printf("Right\n");
    pre_order_print_tree(root->right,level+1);
}
// post-order traversal
void post_order_print_tree(tnode* root,int level){
    if (root==NULL){
        print_format(level);
        printf("...\n");
        return;
    }
    print_format(level);
    printf("%d(L-%d)\n",root->value,level);
    print_format(level);
    printf("Right\n");
    post_order_print_tree(root->right,level+1);
    print_format(level);
    printf(" Left\n");
    post_order_print_tree(root->left,level+1);

}

int main(){
    int choice=1,value;
   tnode* root=NULL;
   while (1){
        if (choice==1){
            printf("\nEnter the value : ");
            scanf("%d",&value);
            root=insert(root,value);
            printf("\n PreOrder view \n");
            pre_order_print_tree(root,0);
        }
        else if (choice==2){
            system("cls");
            printf("\n PostOrder view \n");
            post_order_print_tree(root,0);
        }
        else{
            return 0;
```

```
199         }
200     printf("\n 1 :Insert another node ");
201     printf("\n 2 :PostOrder view of tree\n   Enter your choice : ");
202     scanf("%d",&choice);
203 }
204
205 return 0;
206 }
```