```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,balancing_factor;
    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;
        new_node->balancing_factor=0;
    }
    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;
}

int getBalance(tnode *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}

void initialize_balancing_factor(tnode* root){

       if(root==NULL){
        return;
       }
       root->balancing_factor=getBalance(root);
       initialize_balancing_factor(root->left);
       initialize_balancing_factor(root->right);




}


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

    // Perform rotation
    x->right = y;
```

```
67      y->left = T2;
68
69      // Update heights
70      y->height = int_max(height(y->left), height(y->right))+1;
71      x->height = int_max(height(x->left), height(x->right))+1;
72
73      // Return new root
74      return x;
75  }
76
77  tnode *leftRotate(tnode *x)
78  {
79      tnode *y = x->right;
80      tnode *T2 = y->left;
81
82      // Perform rotation
83      y->left = x;
84      x->right = T2;
85
86      //  Update heights
87      x->height = int_max(height(x->left), height(x->right))+1;
88      y->height = int_max(height(y->left), height(y->right))+1;
89
90      // Return new root
91      return y;
92  }
93
94
95
96  tnode* insert(tnode* node, int value)
97  {
98      /* 1.  Perform the normal BST insertion */
99      if (node == NULL)
100         return(create_tnode(value));
101
102     if (value < node->value)
103         node->left  = insert(node->left, value);
104     else if (value > node->value)
105         node->right = insert(node->right, value);
106     else // Equal values are not allowed in BST
107         return node;
108
109     /* 2. Update height of this ancestor node */
110     node->height = 1 + int_max(height(node->left),
111                           height(node->right));
112
113     /* 3. Get the balance factor of this ancestor
114           node to check whether this node became
115           unbalanced */
116     int balance = getBalance(node);
117
118     // If this node becomes unbalanced, then
119     // there are 4 cases
120
121     // Left Left Case
122     if (balance > 1 && value < node->left->value)
123         return rightRotate(node);
124
125     // Right Right Case
126     if (balance < -1 && value > node->right->value)
127         return leftRotate(node);
128
129     // Left Right Case
130     if (balance > 1 && value > node->left->value)
131     {
132         node->left =  leftRotate(node->left);
```

```
        return rightRotate(node);
    }

    // Right Left Case
    if (balance < -1 && value < node->right->value)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    /* return the (unchanged) node pointer */
    return node;
}

//functions to print a tree (AVL tree)
//this delay function will helps to view the output data properly
void delay(unsigned int mseconds)
{
    clock_t goal = mseconds + clock();
    while (goal > clock());
}

void print_format(int num_of_char){
        delay(25);
        printf("%c",219);
    for(int i=0;i<num_of_char;i++){
        delay(50);
        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);
}
// this function will print the AVL tree with balancing factor(pre-order traversal)
void pre_order_print_with_bf(tnode* root,int level){
    if (root==NULL){
        print_format(level);
        printf("...\n");
        return;
    }
    print_format(level);
    printf("%d(L:%d)(BF:%d)\n",root->value,level,root->balancing_factor);
    print_format(level);
    printf(" Left\n");
    pre_order_print_with_bf(root->left,level+1);
    print_format(level);
    printf("Right\n");
    pre_order_print_with_bf(root->right,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");
             initialize_balancing_factor(root);//this is a recursive function
             printf("\nPrinting the AVL tree with balancing factor \n");
             printf("\nBF : Balancing factor\nL  : level\n");
             pre_order_print_with_bf(root,0);
         }
         else{
             return 0;
         }
     fflush(stdin);//clear buffrer
     printf("\n 1 :Insert another node ");
     printf("\n 2 :initialize balancing factor and print\n   Enter your choice : ");
     scanf("%d",&choice);
}

return 0;
}
```