.vscode\DSA_College\AVL_Tree.cpp

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

struct node{
    public :
    int data ;
    node* left ;
    node* right ;
    int height ;

    // constructor
    node(int d)
    {
        data = d ;
        left = NULL ;
        right = NULL ;
        height = 1 ; // height of leaf node is 1
    }
};

int height(node* root)
{
    if(root == NULL)
    {
        return 0 ;
    }
    int leftH = height(root->left) ;
    int rightH = height(root->right) ;
    return 1 + max(leftH,rightH) ;

    // or simply write, return root->height ;
}

// right rotate subtree rooted with y
node* rightRotate(node* y)
{
    node* x = y->left ;
    node* z = x->right ;

    // perform rotation

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

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

    // return new root
    return x ;
}
```

```
52
53  // left rotate subtree rooted with y
54  node* leftRotate(node* y)
55  {
56      node* x = y->right ;
57      node* z = x->left ;
58
59      // perform rotation
60      x->left = y ;
61      y->right = z ;
62
63      // update heights
64      y->height = 1 + max(height(y->left),height(y->right)) ;
65      x->height = 1 + max(height(x->left),height(x->right)) ;
66
67      // return new root
68      return x ;
69  }
70
71  // get balance factor of node n
72  int getBalance(node* n)
73  {
74      if(n == NULL)
75      return 0 ;
76
77      return height(n->left) - height(n->right) ;
78  }
79
80  // Recursive function to insert a 'd' in
81  // the subtree rooted with 'n'
82  node* insert(node* n, int d)
83  {
84      if(n == NULL)
85      return new node(d) ;
86
87      if(d < n->data)
88      {
89          n->left = insert(n->left,d) ;
90      }
91      else if(d > n->data)
92      {
93          n->right = insert(n->right,d) ;
94      }
95      else{
96          return n ; // equal keys are not allowed in BST
97      }
98
99      int balance = getBalance(n) ;
100
101     // if this node is unbalanced, then there are 4 cases :
102
103     // left left case
104     if(balance > 1 && d < n->left->data)
105     return rightRotate(n) ;
```

```
106
107     // right right rotate
108     if(balance < -1 && d > n->right->data)
109     {
110         return leftRotate(n) ;
111     }
112
113     // left right case
114     if(balance > 1 && d > n->left->data)
115     {
116         n->left = leftRotate(n->left) ;
117         return rightRotate(n) ;
118     }
119
120     // right left case
121     if(balance < -1 && d < n->right->data)
122     {
123         n->right = rightRotate(n->right) ;
124         return leftRotate(n) ;
125     }
126
127     // return the node pointer
128     return n ;
129 }
130
131 void preOrder(node* root)
132 {
133     if (root != nullptr)
134     {
135         cout << root->data << " ";
136         preOrder(root->left);
137         preOrder(root->right);
138     }
139 }
140
141 int main() {
142     node *root = nullptr;
143
144     // Constructing tree given in the above figure
145     root = insert(root, 10);
146     root = insert(root, 20);
147     root = insert(root, 30);
148     root = insert(root, 40);
149     root = insert(root, 50);
150     root = insert(root, 25);
151
152     /* The constructed AVL Tree would be
153               30
154             /   \
155           20     40
156          /  \      \
157         10   25     50
158     */
159     cout << "Preorder traversal : \n";
```

```
    preOrder(root);

    return 0;
}
```