AVL Insertion Implementation

Step 1 :

```
class Node

{

   public:

   int key;

   Node *left;

   Node *right;

   int height;

};
```

Step 2: Function to get maximum of two integers

```
int max(int a, int b)

{

   if (a>b) return a

   else

     return b

}
```

Step 3: Function to get the height of the tree

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

Step 4: Function to create a new node

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

Step 5 : Get Balance factor of node N

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

Step 6 : Function for LL Rotation

```
Node *LL(Node *y)
{
   Node *x = y->left;
   Node *T = x->right;
   x->right = y;
   y->left = T;
   y->height = max(height(y->left),
               height(y->right)) + 1;
   x->height = max(height(x->left),
               height(x->right)) + 1;

   return x;
}
```

Step 7 : Function for RR Rotation

```
  Node *RR(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;
}
```

Step 8 : insert function

```
 Node* insert(Node* node, int key)
{
      /* 1. Find insertion position */
   if (node == NULL)
      return(newNode(key));
     if (key < node->key)
      node->left = insert(node->left, key);
   else if (key > node->key)
      node->right = insert(node->right, key);
   else // Equal keys are not allowed in BST
      return node;
       /* 2. Update height of this ancestor node */
   node->height = 1 + max(height(node->left),
                    height(node->right));
     /* 3. Get the balance factor of this ancestor
      node to check whether this node became
      unbalanced */
   int balance = getBalance(node);
   // If this node becomes unbalanced, then
   // there are 4 cases
```

```
    // Left Left Case
    if (balance > 1 && key < node->left->key)
        return LL(node);

    // Right Right Case
    if (balance < -1 && key > node->right->key)
        return RR(node);

    // Left Right Case
    if (balance > 1 && key > node->left->key)
    {
        node->left = RR(node->left);
        return LL(node);
    }
  // Right Left Case
    if (balance < -1 && key < node->right->key)
    {
        node->right = LL(node->right);
        return RR(node);
    }
   return node;
}
```

Step 9 : Print in ascending order

```
void InOrder(Node *root)
{
    if(root != NULL)
    {
        InOrder(root->left);
        cout << root->key << " ";
        InOrder(root->right);
    }
}
```

Step 10 : Main Function

```
int main()
{
   Node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    cout << "Inorder traversal of the "
            "constructed AVL tree is \n";
    InOrder(root);
      return 0;
}
```