Write a pgm to perform & deletion & insertion operation on AVL trees.

```
class Node
{
  public:
  int key;
  Node *left;
  Node *right;
  int height;
};
int max (int a, int b);
int height (Node *N)
{
    if (N == NULL)
        return 0;
    return N->height;
}
int max (int a, int b)
{
  return (a > b) ? a : b;
}
Node *newNode (int key)
{
  Node *node = new Node();
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
  return (node);
}
```

```
Node *rightRotate(Node *y)
{
  Node *x = y->left;
  Node *T2 = x->right;
  x->right = y;
  y->left = T2;
  y->height = max(height(y->left),
                  height(y->right)) + 1;
  x->height = max(height(x->left),
                  height(x->right)) + 1;
  return x;
}
Node *leftRotate (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 y;
}
```

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

Node* insert(Node* node, int key)
{
  if (node==NULL)
      return(newNode(key));
  if (key < node->key)
      node->left = insert(node->right, key);
  else
    return node;

  node->height = 1+max(height(node->left), height(nod
                                          ->right));

  int balance = getBalance(node);
  if (balance >1 && key < node->left->key)
          return rightRotate(node);

  if (balance <-1 && key > node->right->key)
          return leftRotate(node);

  if (balance >1 && key > node->left->key)
    {
       node->left = leftRotate(node->left);
       return rightRotate(node);
    }
  if (balance <-1 && key < node->right->key)
  { node->right = rightRotate(node->right);
    return leftRotate(node); return node; }
```

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

Node *minValueNode (node * node)
{
    Node * current = node;
    while (current -> left != NULL)
        current = current -> left;
    return current;
}

Node * deleteNode (Node * root, int key)
{
    if (root == NULL)
    return root;
    if (key < root->key)
        root->right = deleteNode (root->right, key);
    else
    {
        if ((root->left == NULL) || (root->right == NULL))
        { Node * temp = root->left ? root->left :
                                     root->right;

    if (temp == NULL)
        { temp = root; root = NULL; }
    else *root = *temp; free (temp); } }
```

```
else
{
  Node * temp = minValueNode (root->right);
  root->key=temp->key;
  root->right= deleteNode (root->right, temp->key);
  }
}

if (root==NULL) return root;
root->height = 1+max(height(root->left), height(root
                                          ->right));

int balance= getBalance(root);
if (balance>1 && getBalance(root->left)>=0)
        return rightRotate(root);

if (balance>1 && getBalance(root->left)<0)
  {  root->left= leftRotate(root->left);
     return rightRotate(root);
  }

if (balance<-1 && getBalance(root->right)<=0)
        return leftRotate(root);
if (balance<-1 && getBalance(root->right)>0)
  {  root->right=rightRotate(root->right);
     return leftRotate(root);
  }
  return root;
}
```