```
else
{
    Node * temp = minValue Node (root -> right);
    root -> key = temp -> key;
    root -> right = delete Node (root -> right,
                                 temp -> key );
  }
}
  if (root == NULL)
  return root;
root -> height = 1 + max (height (root -> left),
                          height (root -> right));
int balance = get Balance (root);

if (balance > 1 && get balance (root -> left)
                >= 0)
    return right Rotate (root);
if (balance > 1 && get balance (root -> left) < 0)
  {
    root -> left = left Rotate (root -> left);
    return right Rotate (root);
  }
if (balance < -1 && get Balance (root -> right)
                 < = 0)
    return left Rotate (root);
if (balance < -1 && get Balance (root -> right) > 0)
  {
    root -> right = right Rotate (root -> right);
    return left Rotate (root);
  }
  return root;
```

## * Deletion

```
Node *deleteNode (Node* root , int key)
{
  if (root == NULL)
     return root;
  if ( key < root -> key )
    root -> left = deleteNode (root->left , key);
  else 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);
    }
```

Rashmi Dhadudi
1BM18CS080

Prog :4 :- Insertion & deletion of AVL tree

* Insertion

```
Node * insert (Node * node, int key)
    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
        return node;
  node -> height = 1 + max (height (node -> left),
                            height (node -> right));
    if (balance > 1 && key < node -> left -> key)
    return right Rotate (node);
  if (balance < -1 && key > node -> right -> key)
        return left Rotate (node);
  if (balance > 1 && key > node -> left -> key)
    {
      node -> left = left Rotate (node -> left);
        return right Rotate (node);
    }
  if (balance < -1 && key < node -> right -> key)
    {
      node -> right = right Rotate (node -> right);
      return left Rotate (node);
    }
        return node;
}
```