We assume each node is a structure with 4 components:
left_child, key, bf, right_child

Algorithm left_balance(root, taller)

Input: root: root node of an avl-tree,
which is out of balance on the
left - the left subtree is too high

On input, root is LH (left high), and its left subtree is
either LH or RH (can not be EH)

taller: boolean indicator, equals true on input

Output: root: new root node of the
updated tree after rotation

taller: equals false on return

Processing: rotate the tree by either single right rotation or
double left-right rotation, dependent on whether
left subtree of root is LH or RH
Update the balance factor (bf) of the relevant nodes
set the taller flag

```
Step1 [prepare]
        child = root->left_child
        taller = false

Step2 [single R rotation?]

        If ( child is LH)
           child -> bf = EH
           root->bf = EH
           right_rotate(root)
           return
         else If (child is EH)
                print error message
                return
Step3 [Double L-R rotation] /* now the child is RH */

        grand_child = child-> right_child
        If (grand_child is LH)
           child->bf = EH
           root->bf = RH
        else if (grand_child is RH)
           child->bf = LH
           root->bf = EH
            else              /* gran_child is EH  - this happens when grand_child is a leaf */
              child->bf = EH
               root->bf =EH

        grand_child->bf = EH      /* bf adjustment finished */

        left_rotate(root->left_child)       /* first rotate left */
        right_rotate(root)       /* then rotate right */
        return
```

****

Left_rotate(root)

Input: root node of an avl-tree which is RH and thus needs to rotate to left.

Output: updated root node after left rotation

Processing: perform the left rotation - after rotation, the right_child of the original root becomes the new root,
the original root node becomes the left_child of the new root.
On return, the pointer previously pointed to old root should
point to the new root.

Step1[get right_child]

```
child = root -> right_child
```

Step2 [exchange pointers]

```
root -> right_child = child -> left_child
      /* the left subtree of child becomes the right subtree of the old root */
child -> left_child = root

     /* the old root now becomes the left_child of the new root */
```

Step3 [fi nish]

```
pointer to root = pointer to child
   /* child is the new root */
return
```