# AVL Tree Functions and Explanations

## 1. getHeight Function

int getHeight(Node\* node) {  
 return node ? node->height : 0;  
}

The `getHeight` function calculates the height of a given node in the AVL tree. If the node is `nullptr`, it returns `0`, as a null node has no height. For non-null nodes, it returns the node's `height` attribute, which represents the height of the subtree rooted at that node.

This function is lightweight and is called frequently during insertions and deletions.

## 2. getBalance Function

int getBalance(Node\* node) {  
 return node ? getHeight(node->left) - getHeight(node->right) : 0;  
}

The `getBalance` function computes the balance factor of a node in the AVL tree. The balance factor is the height difference between the left and right subtrees of a node. If the node is `nullptr`, it returns `0`, indicating no imbalance for a non-existent node.

For any existing node, it calculates the difference in height between the left and right children, which helps to identify if the node is balanced, left-heavy, or right-heavy.

A balance factor of `1`, `0`, or `-1` indicates a balanced node, while values outside this range show imbalance.

This balance factor is essential for deciding whether rotations are necessary to restore AVL tree properties.

The `getBalance` function ensures the AVL tree remains balanced after each modification.

## 3. rightRotate Function

Node\* rightRotate(Node\* y) {

Node\* x = y->left; **// Step 1**

Node\* T2 = x->right; **// Step 2**

x->right = y; **// Step 3**

y->left = T2; **// Step 4**

y->height = max(getHeight(y->left), getHeight(y->right)) + 1; **// Step 5**

x->height = max(getHeight(x->left), getHeight(x->right)) + 1; **// Step 6**

return x; **// Step 7**

}

**Explanation of Each Step**

**Step 1: Node\* x = y->left;**

We identify x, the left child of y, which will become the new root of this rotated subtree. In rotation, x moves up to take the place of y.

**Step 2: Node\* T2 = x->right;**

We temporarily store x’s right subtree (called T2) so that we don’t lose it during the rotation. T2 will later become the left child of y.

**Step 3: x->right = y;**

We make y the right child of x, so x becomes the new root of the subtree, and y shifts down as the right child of x.

**Step 4: y->left = T2;**

y’s left child is updated to T2, which maintains the original structure of the tree by placing T2 as y’s left child.

**Step 5: y->height = max(getHeight(y->left), getHeight(y->right)) + 1;**

We update y’s height based on its new children (left = T2 and right = whatever was originally y’s right subtree).

**Step 6: x->height = max(getHeight(x->left), getHeight(x->right)) + 1;**

We update x’s height as the new root of the subtree with its updated left and right children.

**Step 7: return x;**

We return x as the new root of this subtree after the right rotation. Returning x allows the parent of y to correctly attach this rotated subtree.

## 4. leftRotate Function

Node\* leftRotate(Node\* x) {

Node\* y = x->right; **// Step 1**

Node\* T2 = y->left; **// Step 2**

y->left = x; **// Step 3**

x->right = T2; **// Step 4**

x->height = max(getHeight(x->left), getHeight(x->right)) + 1**; // Step 5**

y->height = max(getHeight(y->left), getHeight(y->right)) + 1; **// Step 6**

return y**; // Step 7**

}

**Explanation of Each Step**

**Step 1: Node\* y = x->right;**

We identify y, the right child of x, which will become the new root of this rotated subtree.

**Step 2: Node\* T2 = y->left;**

We store y’s left subtree (called T2) temporarily so it can be reattached later without getting lost.

**Step 3: y->left = x;**

We make x the left child of y, so y moves up as the new root, and x moves down as the left child of y.

**Step 4: x->right = T2;**

We assign T2 as the right child of x, preserving the structure of the original tree.

**Step 5: x->height = max(getHeight(x->left), getHeight(x->right)) + 1;**

We update x’s height based on its new children after the rotation.

**Step 6: y->height = max(getHeight(y->left), getHeight(y->right)) + 1;**

We update y’s height as the new root of the subtree.

**Step 7: return y;**

We return y as the new root of this subtree after the left rotation.