AVL Trees Implementation (Insertion)

AVL Trees: Implementation

There are **multiple ways** to implement the AVL Trees. We will store the height of each node in the struct, so that we do not have to compute it every time.

```
struct TreeNode
{
    int val;
    TreeNode *left;
    TreeNode *right;
    int height;
};
```

AVL Trees: Implementation

Lets create a class named AVLTree.

```
class AVLTree
{
public:
    TreeNode *root;

    AVLTree()
    {
        root = NULL;
    }
```

AVL Trees: Implementation

Let's Write the function to create the node of the Tree.

```
class AVLTree
{
public:
    TreeNode *root;

    AVLTree()
    {
        root = NULL;
    }
```

```
TreeNode *createNode(int val)
    {
        TreeNode *node = new TreeNode();
        node->val = val;
        node->left = NULL;
        node->right = NULL;
        node->height = 1;
        return node;
    }
```

AVL Trees: Implementation

Let's Write the function to return the Height of the node.

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

AVL Trees: Implementation

Let's Write the function to calculate the BalanceFactor of the node.

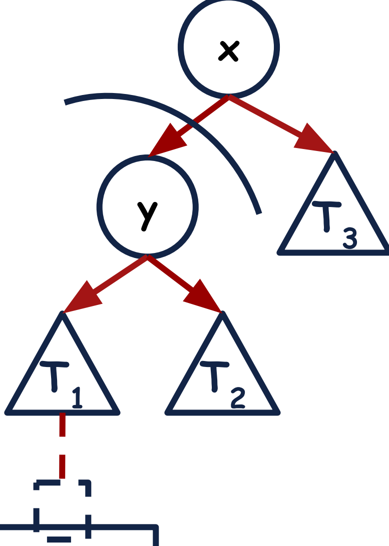
```
int getBalanceFactor(TreeNode *node)
    {
        if (node == NULL)
            return 0;
        return height(node->left) - height(node->right);
    }
```

AVL Trees: Implementation

Let's Write the function to calculate the max of two heights.

```
int max(int a, int b)
    {
        if (a > b)
        {
            return a;
        }
        return b;
    }
```

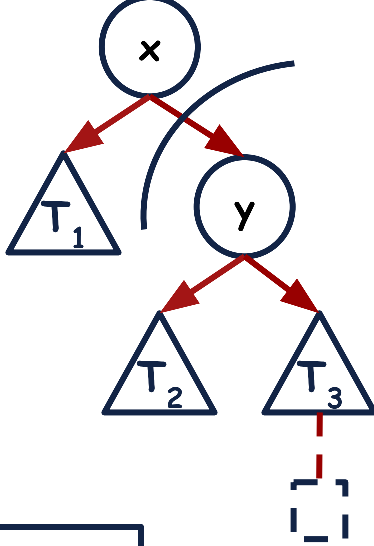
AVL Trees: Implementation



Let's Write the function to rotate Right.

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

AVL Trees: Implementation

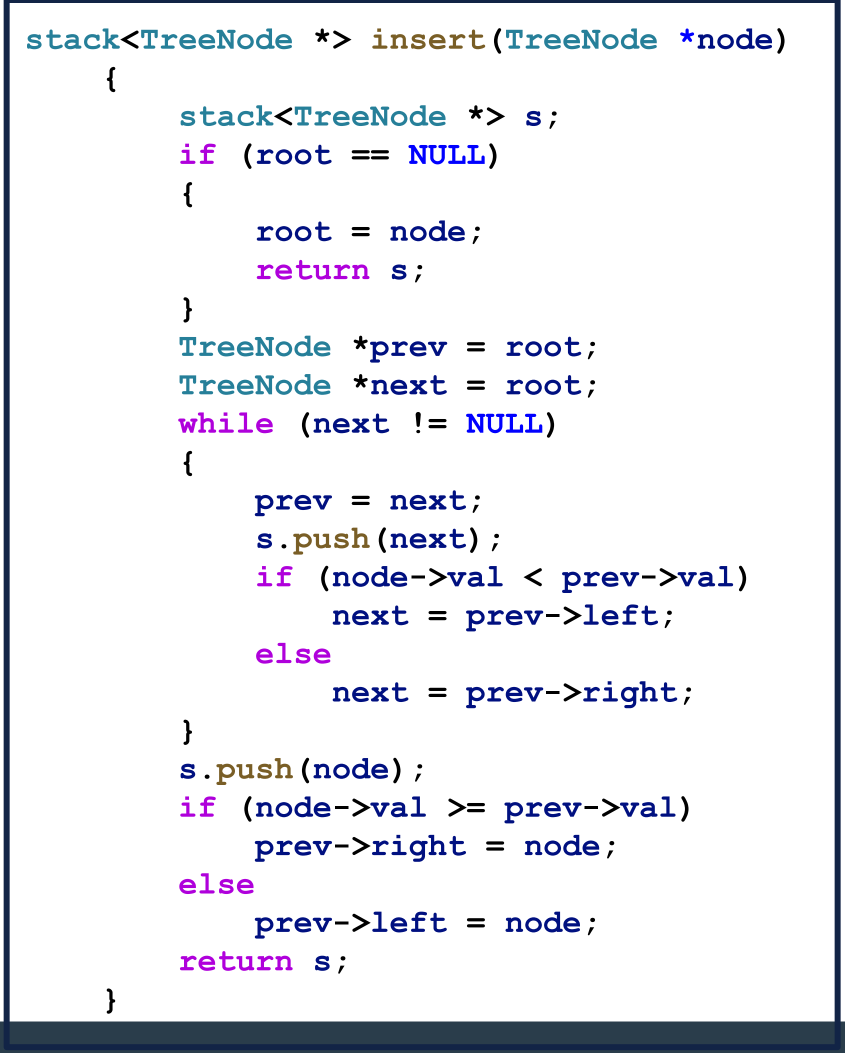


Let's Write the function to rotate Left.

```
TreeNode *leftRotate(TreeNode *x)
    {
        TreeNode *y = x->right;
        TreeNode *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;
    }
```

AVL Trees

Let's Write the function to just insert the node in the tree and store the path it followed to insert the node in the stack.



```
stack<TreeNode *> insert(TreeNode *node)
    {
        stack<TreeNode *> s;
        if (root == NULL)
        {
            root = node;
            return s;
        }
        TreeNode *prev = root;
        TreeNode *next = root;
        while (next != NULL)
        {
            prev = next;
            s.push(next);
            if (node->val < prev->val)
                next = prev->left;
            else
                next = prev->right;
        }
        s.push(node);
        if (node->val >= prev->val)
            prev->right = node;
        else
            prev->left = node;
        return s;
    }
```

AVL Trees

Let's Write the function that we will call for do insertion with rotation.

```
void insertionWithRotation(TreeNode *node)
    {
        stack<TreeNode *> s = insert(node);
        rotate(s, node);
    }
```

```
stack<TreeNode *> insert(TreeNode *node)
    {
        stack<TreeNode *> s;
        if (root == NULL)
        {
            root = node;
            return s;
        }
        TreeNode *prev = root;
        TreeNode *next = root;
        while (next != NULL)
        {
            prev = next;
            s.push(next);
            if (node->val < prev->val)
                next = prev->left;
            else
                next = prev->right;
        }
        s.push(node);
        if (node->val >= prev->val)
            prev->right = node;
        else
            prev->left = node;
        return s;
    }
```

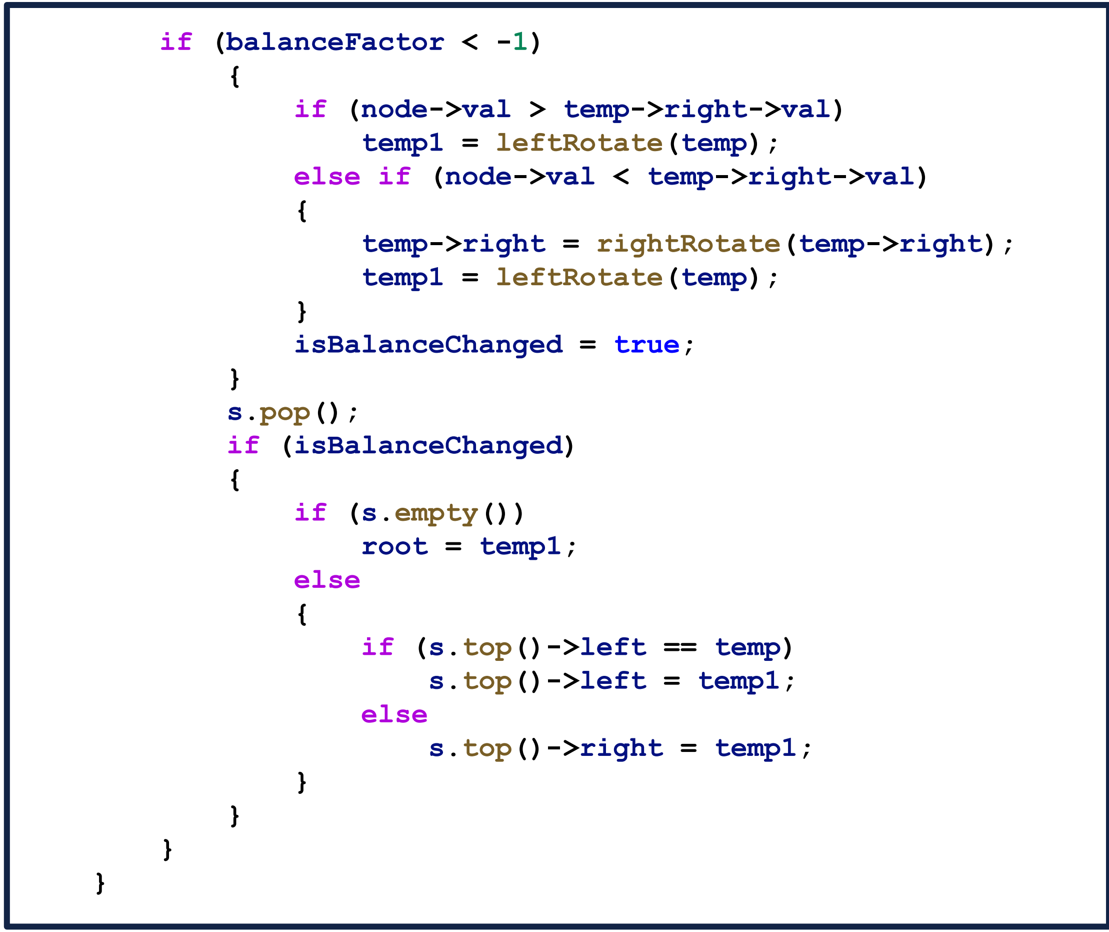
AVL Trees

Let’s Write the function that will do all the heavy work of rotations.

```
void rotate(stack<TreeNode *> s, TreeNode *node)
{
    TreeNode* temp;
    TreeNode *temp1;
    while (!s.empty())
    {
        temp = s.top();
        bool isBalanceChanged = false;
        temp->height = max(height(temp->left), height(temp->right)) + 1;
        int balanceFactor = getBalanceFactor(temp);
        if (balanceFactor > 1)
        {
            if (node->val < temp->left->val)
                temp1 = rightRotate(temp);
            else if (node->val > temp->left->val)
            {
                temp->left = leftRotate(temp->left);
                temp1 = rightRotate(temp);
            }
            isBalanceChanged = true;
        }
```

AVL Trees

Let's Write the function that will do all the heavy work of rotations.



```
    if (balanceFactor < -1)
        {
            if (node->val > temp->right->val)
                temp1 = leftRotate(temp);
            else if (node->val < temp->right->val)
            {
                temp->right = rightRotate(temp->right);
                temp1 = leftRotate(temp);
            }
            isBalanceChanged = true;
        }
        s.pop();
        if (isBalanceChanged)
        {
            if (s.empty())
                root = temp1;
            else
            {
                if (s.top()->left == temp)
                    s.top()->left = temp1;
                else
                    s.top()->right = temp1;
            }
        }
    }
}
```