# Binary Search trees and AVL trees

For this assignment you have to write a program that takes 2 command line arguments as follows:

1. The filename for the input file
2. The filename for the output file

Inside the input file you will have a set of commands and you have to read each command and execute a corresponding function to solve the command

List of function:

* **preorder :** print the tree in preorder -**0p**
* **inorder :** print the tree in inorder -**0p**
* **postorder :** print the tree in postorder -**0p**
* **insert X:** insert a node with value X into the tree (X is a number) while keeping the tree a BST **+10p**
* **delete X:** delete the node which has the value X while keeping the tree a BST **+10p**
* **purge** : deletes and frees the whole tree **+10p**
* **rotate\_right X:** executes rotate\_right on the node with value X **+10p**
* **rotate\_left X:** executes rotate\_left on the node with value X +**10p**
* **insert\_avl X**: inserts a node with value X while keeping the whole tree balanced (called an AVL tree) **+15p**
* **delete\_avl X**: deletes the node with value X while keeping the whole tree balanced (AVL tree). **+20p**
* **warnings (minus) -10p**
* **No mem leaks +10p**
* **No cppcheck warnings/errors +5p**

**You pass if you gather 50 points.**

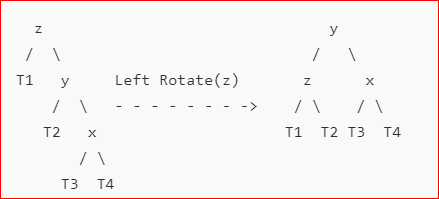
**Example of use cases: Keep in mind you should have the exact format as in the Output file column ( no extra text )**The printing results from the commandsin **bold.**

|  |  |
| --- | --- |
| **Input file** | **Contents of Output file** |
| insert 12  insert 8  insert 11  insert 5  insert 4  insert 18  insert 16  insert 7  insert 2  insert 17  **preorder**  delete 12  **inorder**  insert 6  insert 9  insert 10  delete 8  **postorder**  rotate\_right 9  rotate\_right 16  rotate\_left 16  **preorder**  purge  insert\_avl 12  insert\_avl 8  insert\_avl 11  insert\_avl 5  insert\_avl 4  insert\_avl 18  insert\_avl 16  insert\_avl 7  insert\_avl 2  insert\_avl 17  **preorder**  delete\_avl 4  delete\_avl 2  **postorder** | 12 8 5 4 2 7 11 18 16 17  2 4 5 7 8 11 16 17 18  2 4 6 7 5 10 11 9 17 18 16  5 4 2 18 16 9 11 10 17  12 5 4 2 8 7 11 18 16 17  5 8 7 12 17 16 18 11 |

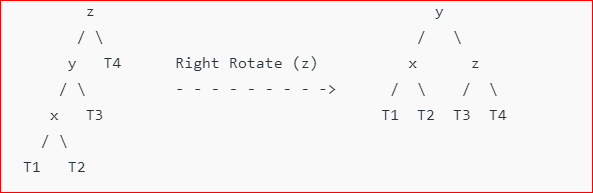
**Support for AVL trees:**

In order to keep the trees balanced we will have to use some rotations:

Left rotation (of z) :



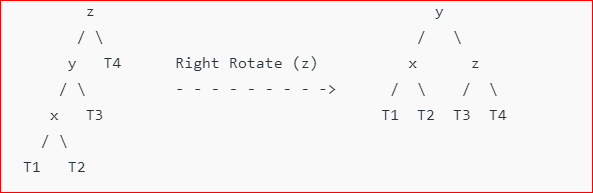
Right rotation (of z):



Note that T1, T2, T3, T4 can be anything (null, a single node, or another tree that is balanced).

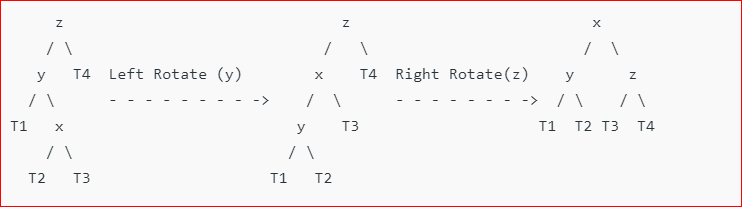
As you can see in the pictures we had some unbalanced trees and using the rotations they became balanced and they BST property still holds for every node. Using these rotations we have 4 cases of unbalanced tree that can be balanced using some rotations.

**Left-Left case (y tree is unbalanced and because of that z is also unbalanced and it is called left-left because the problem nodes, x and y, are both left children):**

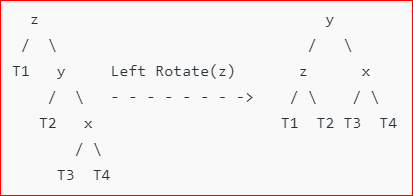


X is the bigger tree between the children of Y, and Y is the bigger child of Z.

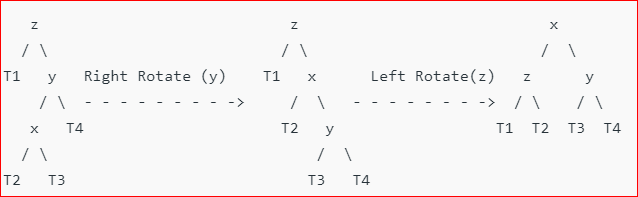
**Left-right case :**



**Right-right case:**



**Right-left case:**



**Steps to follow for insertion**   
Let the newly inserted node be w   
**1)** Perform standard BST insert for w.   
**2)** Starting from w, travel up and find the first unbalanced node(traveling up is done automatically when using recursion). Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.   
**3)** Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:   
a) y is left child of z and x is left child of y (**Left Left** Case)   
b) y is left child of z and x is right child of y (**Left Right** Case)   
c) y is right child of z and x is right child of y (**Right Right** Case)   
d) y is right child of z and x is left child of y (**Right Left** Case)

**Steps to follow for deletion of w:**

**1)** Perform standard BST delete for w.  
**2)** Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the larger height child of z, and x be the larger height child of y. Note that the definitions of x and y are different from [insertion](https://www.geeksforgeeks.org/avl-tree-set-1-insertion/)here.  
**3)** Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:  
a) y is left child of z and x is left child of y (Left Left Case)  
b) y is left child of z and x is right child of y (Left Right Case)  
c) y is right child of z and x is right child of y (Right Right Case)  
d) y is right child of z and x is left child of y (Right Left Case)