# LAB # 11

# Follow the steps to Balance a Binary Search Tree data-structure using rotations in C++ language

## Objective

- To understand about Rotations in Binary Search Tree (BST) and its implementation using C++
- To understand and implement BST Rotation operations using Linked List in C++ language

## Pre-Lab

### What is Balanced Binary Search Tree?

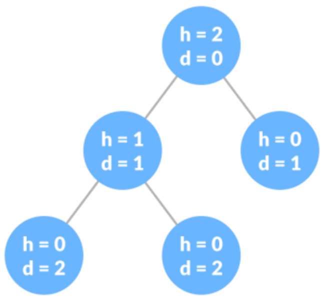
A balanced binary tree, also referred to as a height-balanced binary tree, is defined as a binary tree in which the height of the left and right subtree of any node differs by not more than 1.

### Height of a Node

The height of a node is the number of edges from the node to the deepest leaf (ie. the longest path from the node to a leaf node).

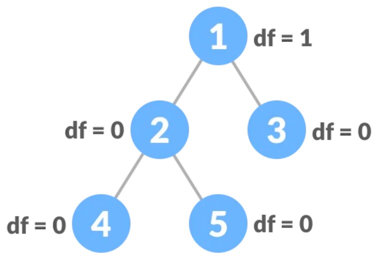
### Height of a Tree

The height of a Tree is the height of the root node or the depth of the deepest node.

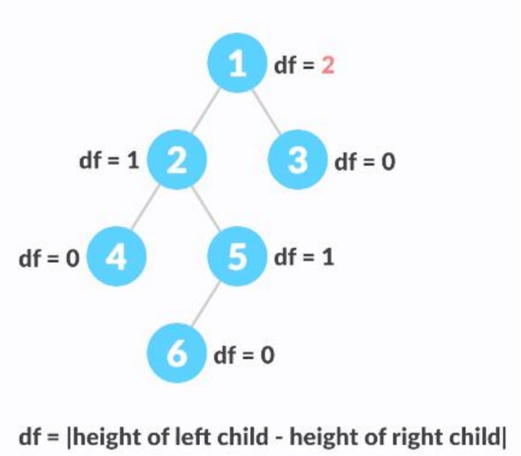



Following are the conditions for a height-balanced binary tree:

i. difference between the left and the right subtree for any node is not more than one
ii. the left subtree is balanced
iii. the right subtree is balanced




*Balanced binary tree with balanced factor*



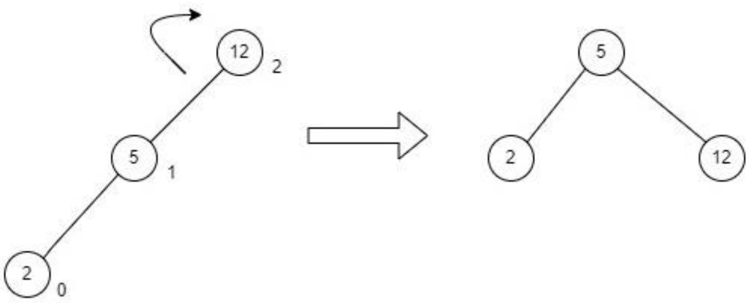

## Rotating the subtrees

There are usually four cases of rotation in the balancing algorithm

i. Left-Left LL
ii. Right-Right RR
iii. Left-Right LR
iv. Right-Left RL.

## Left-Left Rotations (LL)

LL rotation is performed when the node is inserted into the right subtree leading to an unbalanced tree. This is a single left rotation to make the tree balanced again −

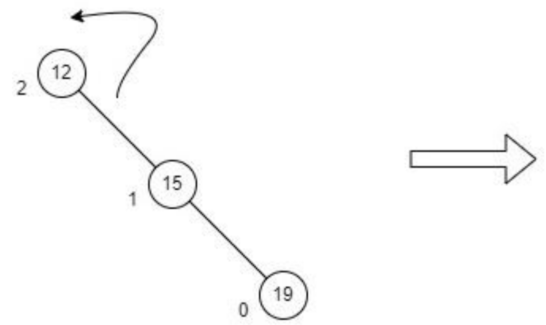



*LL Rotations*

The node where the unbalance occurs becomes the left child and the newly added node becomes the right child with the middle node as the parent node.

## Right-Right Rotations (RR)

RR rotation is performed when the node is inserted into the left subtree leading to an unbalanced tree. This is a single right rotation to make the tree balanced again −



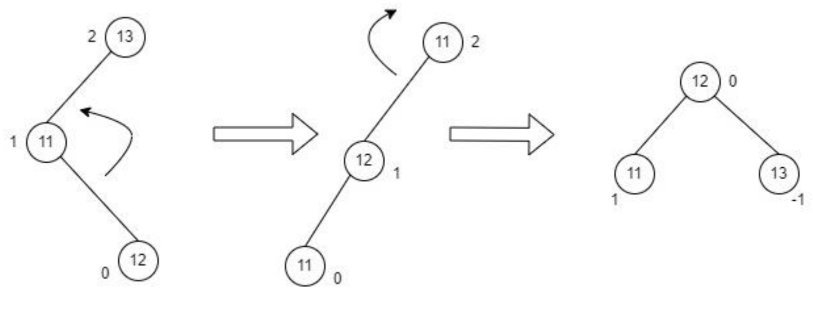

*RR_Rotations*

The node where the unbalance occurs becomes the right child and the newly added node becomes the left child with the middle node as the parent node.

## Left-Right Rotations (LR)

LR rotation is the extended version of the previous single rotations, also called a double rotation. It is performed when a node is inserted into the right subtree of the left subtree. The LR rotation is a combination of the left rotation followed by the right rotation. There are multiple steps to be followed to carry this out.

- Consider an example with “A” as the root node, “B” as the left child of “A” and “C” as the right child of “B”.
- Since the unbalance occurs at A, a left rotation is applied on the child nodes of A, i.e. B and C.
- After the rotation, the C node becomes the left child of A and B becomes the left child of C.
- The unbalance still persists, therefore a right rotation is applied at the root node A and the left child C.
- After the final right rotation, C becomes the root node, A becomes the right child and B is the left child.

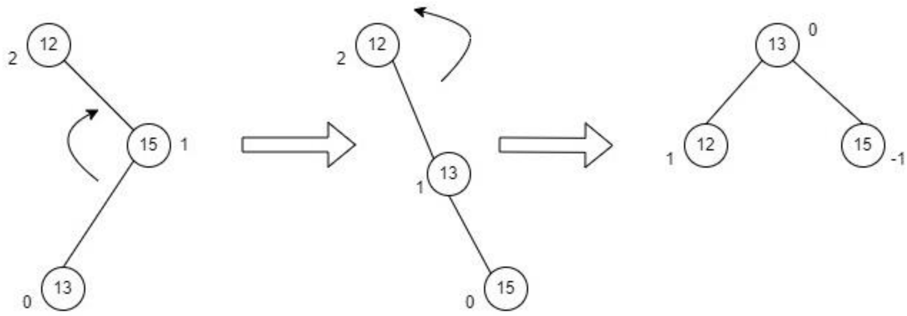



*LR Rotation*

## Right-Left Rotations (RL)

RL rotation is also the extended version of the previous single rotations, hence it is called a double rotation and it is performed if a node is inserted into the left subtree of the right subtree. The RL rotation is a combination of the right rotation followed by the left rotation. There are multiple steps to be followed to carry this out.

- Consider an example with "A" as the root node, "B" as the right child of "A" and "C" as the left child of "B".
- Since the unbalance occurs at A, a right rotation is applied on the child nodes of A, i.e. B and C.
- After the rotation, the C node becomes the right child of A and B becomes the right child of C.
- The unbalance still persists, therefore a left rotation is applied at the root node A and the right child C.
- After the final left rotation, C becomes the root node, A becomes the left child and B is the right child.




*RL Rotations*

## Pre-Lab Task

## Task 1: Follow the procedure to check if a binary tree is height balanced in C++

```
#include <iostream>
#include <stdlib.h>
using namespace std;
#define bool int

// Node creation
struct node {
  int item;
  struct node *left;
  struct node *right;
};

// Create a new node
struct node *newNode(int item) {
  struct node *node = (struct node *)malloc(sizeof(struct
node));
  node->item = item;
  node->left = NULL;
  node->right = NULL;

  return (node);
```

```
}

// Check for height balance
bool checkHeightBalance(struct node *root, int *height) {
  // Check for emptiness
  int leftHeight = 0, rightHeight = 0;
  int l = 0, r = 0;

  if (root == NULL) {
    *height = 0;
    return 1;
  }

  l = checkHeightBalance(root->left, &leftHeight);
  r = checkHeightBalance(root->right, &rightHeight);

*height = (leftHeight > rightHeight ? leftHeight :
rightHeight) + 1;

if ((leftHeight - rightHeight >= 2) || (rightHeight -
leftHeight >= 2))
    return 0;

  else
    return l && r;
}
int main() {
  int height = 0;

  struct node *root = newNode(1);
  root->left = newNode(2);
  root->right = newNode(3);
  root->left->left = newNode(4);
  root->left->right = newNode(5);

  if (checkHeightBalance(root, &height))
    cout<<"The tree is balanced";
  else
    cout<<"The tree is not balanced";
}
```

## Task2: Follow the procedure to rotate a binary tree in C++

```
#include <iostream>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *leftChild;
   struct Node *rightChild;
   int height;
};
int max(int a, int b);
int height(struct Node *N){
   if (N == NULL)
```

```
      return 0;
   return N->height;
}
int max(int a, int b){
   return (a > b) ? a : b;
}
struct Node *newNode(int data){
   struct Node *node = (struct Node *) malloc(sizeof(struct Node));
   node->data = data;
   node->leftChild = NULL;
   node->rightChild = NULL;
   node->height = 1;
   return (node);
}
struct Node *rightRotate(struct Node *y){
   struct Node *x = y->leftChild;
   struct Node *T2 = x->rightChild;
   x->rightChild = y;
   y->leftChild = T2;
   y->height = max(height(y->leftChild), height(y->rightChild)) +
1;
   x->height = max(height(x->leftChild), height(x->rightChild)) +
1;
   return x;
}
struct Node *leftRotate(struct Node *x){
   struct Node *y = x->rightChild;
   struct Node *T2 = y->leftChild;
   y->leftChild = x;
   x->rightChild = T2;
   x->height = max(height(x->leftChild), height(x->rightChild)) +
1;
   y->height = max(height(y->leftChild), height(y->rightChild)) +
1;
   return y;
}
int getBalance(struct Node *N){
   if (N == NULL)
      return 0;
   return height(N->leftChild) - height(N->rightChild);
}
struct Node *minValueNode(struct Node *node){
   struct Node *current = node;
   while (current->leftChild != NULL)
      current = current->leftChild;
   return current;
}
```

## In-Lab Tasks

For all in-lab tasks you are required to

1) Create function of all the algorithms and place them in single header file
2) Use proper “prompts” to tell the user what the program is doing, like printing the elements of array before asking for the value to be searched.
3) Assume the arrays are sorted, especially required for binary and interpolation search.

4) Try to keep your code neat and clean, don't use too many variables when work can be done with lesser number of variables.

## Lab Task 1: Write a C++ program that can provide the mean to check the validity of pre-lab task

Write a C++ code, that provide you with the in-order, pre-order and post-order tree traversal for the BST (use previously created functions from Lab 9). Sketch the tree diagrams and verify the working of pre-lab code.

## Lab Task2: Write a C++ program that rotates an unbalanced tree as per the user's selection

Using pre-lab task 1 and 2, write a C++ program where you will check all four rotations (LL, RR, LR and RL) on a user input tree. Your task is to

- build a user defined BST tree
- print its expressions and draw the tree on paper
- rotate the tree accordingly
- print its expressions and draw the resultant tree on paper
- You should use functions wherever needed
- Your code must be menu driven (code only exits when -1 is entered as input)

## Rubric for Lab Assessment

| **The student performance for the assigned task during the lab session was:** | | | |
|---|---|---|---|
| Excellent | The student completed assigned tasks without any help from the instructor and showed the results appropriately. | 4 | |
| Good | The student completed assigned tasks with minimal help from the instructor and showed the results appropriately. | 3 | |
| Average | The student could not complete all assigned tasks and showed partial results. | 2 | |
| Worst | The student did not complete assigned tasks. | 1 | |

**Instructor Signature: _______________________ Date:________________**