AVL tree insertion and deletion.

// Insertion is first done using BST algorithm after insertion we check the balance factor of every node, and if BF is not right, rotations are performed (LL, RR, RL, LR).

```
struct node * insert(struct node *r, int data)
{     if (r==NULL){          // if ~~tree is empty~~ root is null
          struct node *n;
          n = new struct node;
          n -> data = data.
          r=n
          r->left = r-> ~~lie~~ right = NULL;
     } return r;
     else { if (data< r->data) {.     // if element is less insert to left
               r-> left = insert(r->left, data);
          }
          else {.                    // if greater insert to right
               r-> right = insert(r-> right, data)
          }
     }

// now we check balance factor
r -> height = ~~cal~~height          // for calculating height
if (bf(r) == 2 && bf(r->left) ==1){
          r = llrotation(r); }
else if (bf(r) = -2) && bf(r->right) == -1){
          r = rrrotation(r);
}
else if(bf(r) == -2 && bf(r->right) == -1){
          r = rlrotation(r);
}
else if (bf(r) == 2 && bf(r->left) == -1){
}         r= lrrotation(r);
 return r   }                        // insertion complete
```

// Deletion.

- Check for element in the AVL tree
- delete the node how we do in BST.
- and now do rotation for setting the balance factor right

```
struct node * deletenode(struct node *p, int
                                        data)
{
      if (p -> left = =NULL && p -> right = NULL)
     {
             if (p = = this -> root)
                    this -> root =NULL
             delete p;
             return NULL;
      }
      if (p -> data < data)       //then data is present in
                                                       right.
     {
        p -> right = delete Node(p -> right, data);
     }
     else if (p -> data > data) // then data is present
                                                 in left
     {
        p -> left = delete Node(p -> right, data)
     }
     else {    if (p -> left != NULL) {        // if ~~for a~~ left
                                                or right tree is
                                                       present
               ~~q = in~~
               p -> data = q -> data;
               p -> left = deleteNode(p -> left, q -> data);
          }
          else {
               q = insucc (p -> right);
               p -> data = q -> data;
               p -> right = deleteNode(p -> right, q -> data);
     }   }
     if (bf(p) = = 2 && (bf(p -> left) = = 1) { p = LLrotation(p);
else if (bf(p) ==2 && bf(p->left) == -1){
            p = lrrotation(p); }
else if (bf(p) == 2 && bf (p->left ==0) {
            p = llrotation }
else if (bf(p) = = -2 && bf (p->right) == -1){
        p = rrrotation(p);}
else if (bf(p) = = -2 && bf (p->right) ==1){
            p = rlrotation(p);
else if(bf(p) == -2&&bf(p->right ==0){
                p = llrotation(p); }
    }
    return p;
}
```

R. All four rotations are called in above fuction should be implemented in following manner.

```
struct rrrotation(struct node *n ){
            struct node *p.
            struct node *tp;
            p = n;
            tp = p->right;
            p->right = tp->left;
            tp->left =p;
            return tp;
    }
```