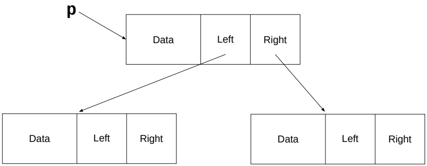
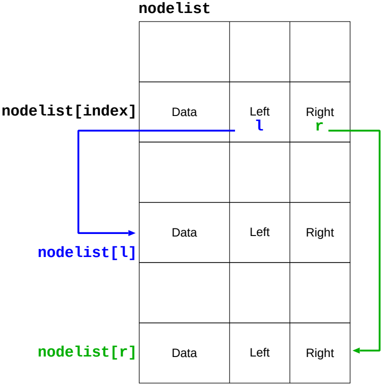
# AVL Trees

Data and File Structures Laboratory

http://www.isical.ac.in/~dfslab/2018/index.html

# Recap: traditional vs. alternative implementations




```
NODE *p;
p->data
```

`root` (type: `NODE *p`)

```
int index;
tree->nodelist[index].data
```

`tree->root` (type: `int`)

# Type definitions

```
typedef int DATA;

typedef struct node {
    DATA data;
    int left, right, parent, height; // following Weiss, DS
   & AA in C++, 4ed.
} AVL_NODE;

typedef struct {
    unsigned int num_nodes, max_nodes;
    int root, free_list;
    AVL_NODE *nodelist;
} TREE;
```

# API functions

```
extern int init_tree(TREE *);
extern int search(TREE *, int , DATA);
extern int insert(TREE *, int , int *, DATA);
extern int delete(TREE *, int , int *, DATA);

#define DELETE_TREE(tree) free(tree->nodelist);
```

`parent` index (optional)

# Helper functions

```
extern int grow_tree(TREE *);
extern int get_new_node(TREE *);
extern void free_up_node(TREE *, int);
extern int find_successor(TREE *, int);
extern void rotate_on_insert_LL(TREE *, int , int *);
extern void rotate_on_insert_RR(TREE *, int , int *);
extern void rotate_on_insert_LR(TREE *, int , int *);
extern void rotate_on_insert_RL(TREE *, int , int *);
extern void balance(TREE *, int *);

extern void inorder(TREE *, int);
extern void print_pstree(TREE *, int);

#define HEIGHT(T, nodeindex) ( ((nodeindex) == -1) ? -1 :
    T->nodelist[nodeindex].height )
```

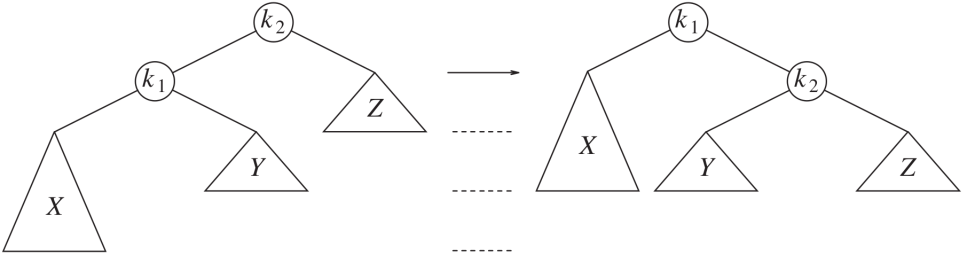
# find_successor() I

```
int find_successor(TREE *tree, int node) {
    int child;
    assert(node != -1);
    /* Go to right child, then as far left as possible */
    child = tree->nodelist[node].right;
    if (child == -1) /* no successors */
        return -1;
    if (tree->nodelist[child].left == -1) {
        /* Don't do this here for AVL trees */
        /* tree->nodelist[node].right = tree->nodelist[child].right; */
        /* if (tree->nodelist[child].right != -1) */
        /*     tree->nodelist[tree->nodelist[child].right].parent =
    node; */
        return child;
    }
    while (tree->nodelist[child].left != -1) {
        node = child;
        child = tree->nodelist[child].left;
```

# find_successor() II

```
    }
    /* Don't do this here for AVL trees */
    /* tree->nodelist[node].left = tree->nodelist[child].right; */
    /* if (tree->nodelist[child].right != -1) */
    /*     tree->nodelist[tree->nodelist[child].right].parent = node; */
    return child;
}
```

# Rotate LL



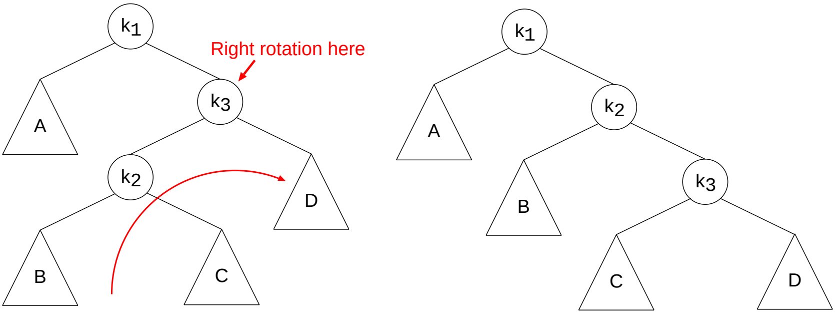
# Rotate LL I

```
void rotate_on_insert_LL(TREE *tree, int parent, int *node) {
    /* See Weiss, DS & AA in C++, 4 ed., Section 4.4.1, Figure 4.34 */
    int k2 = *node;
    int k1 = tree->nodelist[k2].left;
    int Z = tree->nodelist[k2].right;
    int X = tree->nodelist[k1].left;
    int Y = tree->nodelist[k1].right;

    /* rotate */
    tree->nodelist[k2].left = Y;
    tree->nodelist[k1].right = k2;

    /* parents (optional) */
    tree->nodelist[k1].parent = parent;
    tree->nodelist[k2].parent = k1;
    if (Y != -1) tree->nodelist[Y].parent = k2;
```

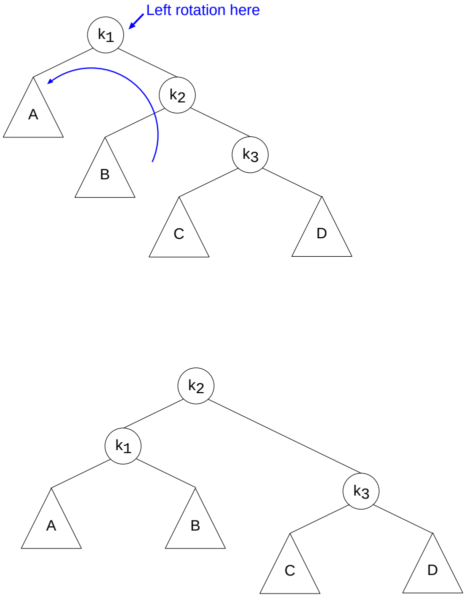
# Rotate LL II

```
    /* update heights */
    tree->nodelist[k2].height = 1 +
        MAX(tree->nodelist[Y].height, tree->nodelist[Z].height);
    tree->nodelist[k1].height = 1 +
        MAX(tree->nodelist[X].height, tree->nodelist[k2].height);

    *node = k1;
    return;
}
```

# Rotate RL



# Rotate RL



# Rotate RL

```
void rotate_on_insert_RL(TREE *tree, int parent, int *node) {
    /* See CMSC 420 Lecture Notes by David M. Mount, UMCP, pg. 39. */
    int k1 = *node;
    rotate_on_insert_RR(tree, k1, &(tree->nodelist[k1].right));
    rotate_on_insert_LL(tree, parent, node);
    return;
}
```

# Problems I

1. Complete the missing parts (marked `TODO`) in the provided implementation of AVL trees.
2. Given a tree that is supposed to be an AVL tree, write a program to check whether it is indeed an AVL tree, and whether all fields have correct / consistent values.

   You will need to check whether the `left`, `right`, and `parent` fields are consistent, whether the `height` field is correct (if not, fill in the field with the correct value), and finally whether imbalances (if any) are within the permissible limit.

   The tree will be given to you via an input file, using a format similar to the one used in Lab Test 2. The name of the input file will be given as a command-line argument.

# Problems II

3. Use program 2 to test (and debug if necessary) the AVL tree implementation in program 1.

   Note that the provided implementation has not been tested; it is therefore likely to have bugs (in addition to being incomplete). Please let me know about existing bugs so that I can fix them. Sorry for being lazy!