Name – Shrinivas Hatyalikar
Div - CS-B
Roll no.- 24
PRN- 12110883

WAP to Implement AVL tree and display it levelwise.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct Node
{
   struct Node *lchild;
   int data;
   int bf;
   struct Node *rchild;
} *root = NULL;
int height(struct Node *p)
{
   int x = 0, y = 0;
   if (!p)
      return 0;
   x = height(p->lchild);
   y = height(p->rchild);
   return x > y ? x + 1 : y + 1;
}
void Insert(int key)
{
   struct Node *t = root;
   struct Node *r = NULL, *p;

   if (root == NULL)
   {
      p = (struct Node *)malloc(sizeof(struct Node));
      p->data = key;
```

```
        p->bf = 0;
        p->lchild = p->rchild = NULL;
        root = p;
        return;
    }
    while (t != NULL)
    {
        r = t;
        if (key < t->data)
            t = t->lchild;
        else if (key > t->data)
            t = t->rchild;
        else
            return;
    }
    p = (struct Node *)malloc(sizeof(struct Node));
    p->data = key;
    p->lchild = p->rchild = NULL;

    if (key < r->data)
        r->lchild = p;
    else
        r->rchild = p;
}
struct Node *LLRotation(struct Node *p)
{
    int lbf, rbf;
    struct Node *pl = p->lchild;
    pl->bf = 0;
    p->lchild = pl->rchild;
    pl->rchild = p;
    lbf = height(p->lchild) + 1;
    rbf = height(p->rchild) + 1;
    p->bf = lbf - rbf;
    if (p == root)
        root = pl;
    return pl;
```

```
}
struct Node *LRRotation(struct Node *p)
{
   int lbf, rbf;
   struct Node *pl = p->lchild;
   struct Node *plr = pl->rchild;
   plr->bf = 0;

   p->lchild = plr->rchild;
   pl->rchild = plr->lchild;
   plr->lchild = pl;
   plr->rchild = p;
   lbf = height(p->lchild) + 1;
   rbf = height(p->rchild) + 1;
   p->bf = lbf - rbf;

   lbf = height(pl->lchild) + 1;
   rbf = height(pl->rchild) + 1;
   pl->bf = lbf - rbf;
   if (p == root)
      root = plr;
   return plr;
}
struct Node *RRRotation(struct Node *p)
{
   int lbf, rbf;
   struct Node *pr = p->rchild;
   pr->bf = 0;
   p->rchild = pr->lchild;
   pr->lchild = p;
   lbf = height(p->lchild) + 1;
   rbf = height(p->rchild) + 1;
   p->bf = lbf - rbf;
   if (p == root)
      root = pr;
   return pr;
}
```

```
struct Node *RLRotation(struct Node *p)
{
   int lbf, rbf;
   struct Node *pr = p->rchild;
   struct Node *prl = pr->lchild;
   prl->bf = 0;

   p->rchild = prl->lchild;
   pr->lchild = prl->rchild;
   prl->rchild = pr;
   prl->lchild = p;
   lbf = height(p->lchild) + 1;
   rbf = height(p->rchild) + 1;
   p->bf = lbf - rbf;

   lbf = height(pr->lchild) + 1;
   rbf = height(pr->rchild) + 1;
   pr->bf = lbf - rbf;
   if (p == root)
      root = prl;
   return prl;
}
struct Node *RInsert(struct Node *p, int key)
{
   struct Node *t;
   int lbf, rbf;
   if (p == NULL)
   {
      t = (struct Node *)malloc(sizeof(struct Node));
      t->data = key;
      t->bf = 0;
      t->lchild = t->rchild = NULL;
      return t;
   }
   if (key < p->data)
      p->lchild = RInsert(p->lchild, key);
   else if (key > p->data)
```

```
        p->rchild = RInsert(p->rchild, key);

    lbf = height(p->lchild) + 1;
    rbf = height(p->rchild) + 1;
    p->bf = lbf - rbf;
    if (p->bf == 2 && p->lchild->bf == 1)
        return LLRotation(p);
    if (p->bf == 2 && p->lchild->bf == -1)
        return LRRotation(p);
    if (p->bf == -2 && p->rchild->bf == -1)
        return RRRotation(p);
    if (p->bf == -2 && p->rchild->bf == 1)
        return RLRotation(p);
    return p;
}
void Inorder(struct Node *p)
{
    if (p)
    {
        Inorder(p->lchild);
        printf("%d ", p->data);
        Inorder(p->rchild);
    }
}
void LevelOrder(struct Node *root) {
    if (root == NULL) {
        return;
    }
    struct Node *queue[100];
    int front = -1, rear = -1;
    queue[++rear] = root;

    while (front != rear) {
        struct Node *temp = queue[++front];
        printf("%d ", temp->data);
        if (temp->lchild != NULL) {
            queue[++rear] = temp->lchild;
```

```
        }
        if (temp->rchild != NULL) {
            queue[++rear] = temp->rchild;
        }
    }
}

// struct Node *Search(int key)
// {
//     struct Node *t = root;

//     while (t != NULL)
//     {
//         if (key == t->data)
//             return t;
//         else if (key < t->data)
//             t = t->lchild;
//         else
//             t = t->rchild;
//     }
//     return NULL;
// }
int main()
{
    //struct Node *temp;

    Insert(30);
    RInsert(root, 10);
    RInsert(root, 15);
    RInsert(root, 20);
    RInsert(root, 45);
    RInsert(root, 50);
    RInsert(root, 5);
    printf("\nInorder :\n");
    Inorder(root);
    printf("\nLevel Order: \n");
    LevelOrder(root);
```

```
    printf("\n");
    // temp = Search(2);
    // if (temp != NULL)
    //     printf("element %d is found\n", temp->data);
    // else
    //     printf("element is not found\n");
    return 0;
}
```

Output:

```
Inorder :
5 10 15 20 30 45 50
Level Order:
30 15 45 10 20 50 5
```