WEEK 10 - Implementation of AVL Tree

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
   int key;
   struct Node *left;
   struct Node *right;
   int height;
};
typedef struct Node node;
int height(node *n)
{
   if (n==NULL)
      return 0;
   return n->height;
}
node *findmin(node *tree)
{ if(tree==NULL)
  return NULL;
 else if(tree->left==NULL)
    return tree;
  else
    return findmin(tree->left);
}

int max(int a,int b)
```

```
{ return (a>b)?a:b;
}
node *rightrotate(node *y)
{ node *x=y->left;
  node *t2=x->right;

  x->right=y;
  y->left=t2;
  y->height=1+max(height(y->left),height(y->right));
  x->height=1+max(height(x->left),height(x->right));
  return x;
}

node *leftrotate(node *x)
{ node *y=x->right;
  node *t2=y->left;

  y->left=x;
  x->right=t2;
  x->height=1+max(height(x->left),height(x->right));
  y->height=1+max(height(y->left),height(y->right));
  return y;
}
int getbalance(struct Node *n)
{
    if (n == NULL)
        return 0;
    return height(n->left) - height(n->right);
}
```

```
node *insert(node *tree,int k)
{ if(tree==NULL)
  { node *newnode=malloc(sizeof(node));
    newnode->key=k;
    newnode->left=NULL;
    newnode->right=NULL;
    newnode->height=1;
    tree=newnode;
  }

  else if(k<tree->key)
      tree->left=insert(tree->left,k);
  else if(k>tree->key)
      tree->right=insert(tree->right,k);
  //else
     //return tree;
  tree->height=1+max(height(tree->left),height(tree->right));
  int bal=getbalance(tree);
  if(bal>1 && k<tree->left->key)
       return rightrotate(tree);
  if(bal<-1 && k>tree->right->key)
       return leftrotate(tree);
  if(bal>1 && k>tree->left->key)
  {    tree->left=leftrotate(tree->left);
       return rightrotate(tree);  }
  if(bal<-1 && k<tree->right->key)
  {    tree->right=rightrotate(tree->right);
       return leftrotate(tree);  }
  return tree;
}
```

```
node *delete(node *tree,int e)
{  node *temp=malloc(sizeof(node));
  if(e<tree->key)
    tree->left=delete(tree->left,e);
  else if(e>tree->key)
    tree->right=delete(tree->right,e);
  else if(tree->left && tree->right)
   { temp=findmin(tree->right);
     tree->key=temp->key;
     tree->right=delete(tree->right,temp->key);
   }
  else
  { temp=tree;
   if(tree->left==NULL)
      tree=tree->right;
   else if(tree->right==NULL)
      tree=tree->left;
   free(temp);
  }
  if (tree == NULL)
      return tree;

  tree->height = 1 + max(height(tree->left),
                         height(tree->right));

  int balance = getbalance(tree);

   if (balance > 1 &&
      getbalance(tree->left) >= 0)
      return rightrotate(tree);
```

```
    // Left Right Case
    if (balance > 1 &&
        getbalance(tree->left) < 0)
    {
        tree->left = leftrotate(tree->left);
        return rightrotate(tree);
    }

    // Right Right Case
    if (balance < -1 &&
        getbalance(tree->right) <= 0)
        return leftrotate(tree);

    // Right Left Case
    if (balance < -1 &&
        getbalance(tree->right) > 0)
    {
        tree->right = rightrotate(tree->right);
        return leftrotate(tree);
    }

    return tree;
}
void inorder(node *tree)
{ if(tree!=NULL)
  { //printf("%d ",tree->key);
    inorder(tree->left);
    printf("%d ",tree->key);
    inorder(tree->right);}
}
```

```
int main()
{
   node *tree=NULL;
   int n;
   printf("ENTER TOTAL NUMBER OF ELEMENTS");
   scanf("%d",&n);
   int e;
   printf("ENTER ELEMENTS");
   for(int i=0;i<n;i++)
   {    scanf("%d",&e);
      tree=insert(tree,e);}
   //inorder(tree);
   printf("ENTER ELEMENT TO BE DELETED");
   scanf("%d",&e);
   tree = delete(tree,e);
   inorder(tree);



   return 0;
}
```

OUTPUT:-

```
 ENTER TOTAL NUMBER OF ELEMENTS 9
ENTER ELEMENTS 9 5 10 0 6 11 -1 1 2
ENTER ELEMENT TO BE DELETED 10
-1 0 1 2 5 6 9 11
```