DATASTRUCTURES

31/7/24

1.write a c programe for avl programe in insertion,deletion and search.

```
#include <stdio.h>
#include <stdlib.h>
 typedef struct AVLNode
{
   int key;
   struct AVLNode* left;
   struct AVLNode* right;
   int height;
}
AVLNode;
int height(AVLNode* node)
{
   return node ? node->height : 0;
}
AVLNode* newNode(int key)
{
   AVLNode* node = (AVLNode*)malloc(sizeof(AVLNode));
   node->key = key;
   node->left = NULL;
   node->right = NULL;
   node->height = 1;
   return node;
```

```
}
int getBalance(AVLNode* node)
 {
   return node ? height(node->left) - height(node->right) : 0;
}
AVLNode* rightRotate(AVLNode* y)
{
   AVLNode* x = y->left;
   AVLNode* T2 = x->right;

   x->right = y;
   y->left = T2;

   y->height = 1 + (height(y->left) > height(y->right) ? height(y->left) : height(y->right));
   x->height = 1 + (height(x->left) > height(x->right) ? height(x->left) : height(x->right));

   return x;
}
AVLNode* leftRotate(AVLNode* x)
 {
   AVLNode* y = x->right;
   AVLNode* T2 = y->left;
   y->left = x;
   x->right = T2;

   x->height = 1 + (height(x->left) > height(x->right) ? height(x->left) : height(x->right));
```

```
    y->height = 1 + (height(y->left) > height(y->right) ? height(y->left) : height(y->right));


    return y;
}
AVLNode* insert(AVLNode* node, int key)
 {
    if (node == NULL)
        return newNode(key);


    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        return node;
    node->height = 1 + (height(node->left) > height(node->right) ? height(node->left) :
height(node->right));


    int balance = getBalance(node);
     if (balance > 1 && key < node->left->key)
        return rightRotate(node);
    if (balance > 1 && key > node->left->key)
{
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
```

```
    if (balance < -1 && key > node->right->key)
        return leftRotate(node);
    if (balance < -1 && key < node->right->key)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}
AVLNode* minValueNode(AVLNode* node)
{
    AVLNode* current = node;
    while (current->left != NULL)
        current = current->left;
    return current;
}
AVLNode* deleteNode(AVLNode* root, int key)
{
    if (root == NULL)
        return root;

    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
```

```
    else
{
        if (root->left == NULL)
{
            AVLNode* temp = root->right;
            free(root);
            return temp;
        }
else if (root->right == NULL)
{
            AVLNode* temp = root->left;
            free(root);
            return temp;
        }

        AVLNode* temp = minValueNode(root->right);
        root->key = temp->key;
        root->right = deleteNode(root->right, temp->key);
    }

    if (root == NULL)
        return root;

    root->height = 1 + (height(root->left) > height(root->right) ? height(root->left) : height(root->right));
```

```
   int balance = getBalance(root);
    if (balance > 1 && getBalance(root->left) >= 0)
      return rightRotate(root);
    if (balance > 1 && getBalance(root->left) < 0)
{
      root->left = leftRotate(root->left);
      return rightRotate(root);
   }
    if (balance < -1 && getBalance(root->right) <= 0)
      return leftRotate(root);
   if (balance < -1 && getBalance(root->right) > 0)
 {
      root->right = rightRotate(root->right);
      return leftRotate(root);
   }

   return root;
}
AVLNode* search(AVLNode* root, int key)
 {
   if (root == NULL || root->key == key)
      return root;

   if (key < root->key)
      return search(root->left, key);
   else
```

```
    return search(root->right, key);
}
void inorder(AVLNode* root)
 {
   if (root != NULL)
 {
    inorder(root->left);
    printf("%d ", root->key);
    inorder(root->right);
   }
}
int main()
 {
   AVLNode* root = NULL;
    root = insert(root, 10);
   root = insert(root, 20);
   root = insert(root, 30);
   root = insert(root, 15);
   root = insert(root, 25);
   root = insert(root, 5);

   printf("Inorder traversal of the AVL tree:\n");
   inorder(root);
   printf("\n");
    int keyToSearch = 15;
   AVLNode* result = search(root, keyToSearch);
```

```
    if (result)
        printf("Node with key %d found.\n", keyToSearch);
    else
        printf("Node with key %d not found.\n", keyToSearch);
     root = deleteNode(root, 10);
    printf("Inorder traversal after deleting 10:\n");
    inorder(root);
    printf("\n");

    return 0;
}
```

OUTPUT:

```
Inorder traversal of the AVL tree:
5 10 15 20 25 30
 Node with key 15 found.
 Inorder traversal after deleting 10:
5 15 20 25 30
```