DATA STRUCTURE

DAY-6

1.AVL program for insertion,deletion and search in c.

Program:

```
#include <stdio.h>
#include <stdlib.h>
 typedef struct AVLNode {
   int key;
   int height;
   struct AVLNode* left;
   struct AVLNode* right;
} AVLNode;
 int height(AVLNode* node) {
   return node ? node->height : 0;
}
 int max(int a, int b) {
   return (a > b) ? a : b;
}
AVLNode* rightRotate(AVLNode* y) {
   AVLNode* x = y->left;
   AVLNode* T2 = x->right;
   x->right = y;
   y->left = T2;
   y->height = max(height(y->left), height(y->right)) + 1;
   x->height = max(height(x->left), height(x->right)) + 1;
   return x;
}
```

```
AVLNode* leftRotate(AVLNode* x) {
  AVLNode* y = x->right;
  AVLNode* T2 = y->left
  y->left = x;
  x->right = T2;
  x->height = max(height(x->left), height(x->right)) + 1;
  y->height = max(height(y->left), height(y->right)) + 1;
  return y;
}
int getBalance(AVLNode* node) {
  return node ? height(node->left) - height(node->right) : 0;
}
AVLNode* insert(AVLNode* node, int key) {
  if (!node) {
    AVLNode* newNode = (AVLNode*)malloc(sizeof(AVLNode));
    newNode->key = key;
    newNode->left = newNode->right = NULL;
    newNode->height = 1;
    return newNode;
  }
  if (key < node->key)
    node->left = insert(node->left, key);
  else if (key > node->key)
    node->right = insert(node->right, key);
  else
    return node; // Duplicate keys are not allowed
  node->height = 1 + max(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->right->key)
        return leftRotate(node);

    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(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)
        current = current->left;
    return current;
}
 AVLNode* deleteNode(AVLNode* root, int key) {
    if (!root)
        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 || !root->right) {
      AVLNode* temp = root->left ? root->left : root->right;
      if (!temp) {
         temp = root;
         root = NULL;
      } else
         *root = *temp;
      free(temp);
   } else {
      AVLNode* temp = minValueNode(root->right);
      root->key = temp->key;
      root->right = deleteNode(root->right, temp->key);
   }
}
if (!root)
   return root;
root->height = 1 + max(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 || root->key == key)
        return root;
    if (key < root->key)
        return search(root->left, key);
    return search(root->right, key);
}
 void inorder(AVLNode* root) {
    if (root) {
        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);
    printf("Inorder traversal of the AVL tree:\n");
    inorder(root);
```

```
    printf("\n");
 AVLNode* result = search(root, 15);
    if (result)
        printf("Key 15 found in the AVL tree.\n");
    else
        printf("Key 15 not found in the AVL tree.\n");
    root = deleteNode(root, 20);
    printf("Inorder traversal after deletion of 20:\n");
    inorder(root);
    printf("\n");
     return 0;
}
```

Output:

Inorder traversal of the AVL tree:

10 15 20 30

Key 15 found in the AVL tree.

Inorder traversal after deletion of 20:

10 15 30