**Assignment 11**

**Name** : Komal Potdar

**Roll No**.: 92

**PRN No**.: 12320165

**Div**: CS B SY

**Batch**: 3

**1.WAP to implement AVL tree for creation and Insertion and Display.**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  typedef struct AVLNode {  int data;  struct AVLNode \*left;  struct AVLNode \*right;  int height;  } AVLNode;  int height(AVLNode \*node) {  if (node == NULL)  return 0;  return node->height;  }  int max(int a, int b) {  return (a > b) ? a : b;  }  AVLNode \*createNode(int data) {  AVLNode \*newNode = (AVLNode \*)malloc(sizeof(AVLNode));  if (newNode == NULL) {  printf("Memory allocation failed!\n");  exit(EXIT\_FAILURE);  }  newNode->data = data;  newNode->left = NULL;  newNode->right = NULL;  newNode->height = 1;  return newNode;  }  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) {  if (node == NULL)  return 0;  return height(node->left) - height(node->right);  }  AVLNode \*insert(AVLNode \*node, int data) {  if (node == NULL)  return createNode(data);  if (data < node->data)  node->left = insert(node->left, data);  else if (data > node->data)  node->right = insert(node->right, data);  else  return node;  node->height = max(height(node->left), height(node->right)) + 1;  int balance = getBalance(node);  if (balance > 1 && data < node->left->data)  return rightRotate(node);  if (balance < -1 && data > node->right->data)  return leftRotate(node);  if (balance > 1 && data > node->left->data) {  node->left = leftRotate(node->left);  return rightRotate(node);  }  if (balance < -1 && data < node->right->data) {  node->right = rightRotate(node->right);  return leftRotate(node);  }  return node;  }  void inorder(AVLNode \*root) {  if (root != NULL) {  inorder(root->left);  printf("%d ", root->data);  inorder(root->right);  }  }  void freeAVLTree(AVLNode \*root) {  if (root != NULL) {  freeAVLTree(root->left);  freeAVLTree(root->right);  free(root);  }  }  int main() {  AVLNode \*root = NULL;  int data, choice;  do {  printf("\n1. Insert\n");  printf("2. Display Inorder Traversal\n");  printf("3. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  switch (choice) {  case 1:  printf("Enter data to insert: ");  scanf("%d", &data);  root = insert(root, data);  break;  case 2:  printf("Inorder traversal of AVL tree: ");  inorder(root);  printf("\n");  break;  case 3:  printf("Exiting...\n");  break;  default:  printf("Invalid choice!\n");  }  } while (choice != 3);  freeAVLTree(root);  return 0;  } |

Output:

