**1.Implement AVL tree**

#include <stdio.h>

#include <stdlib.h>

struct AVLNode {

int key;

struct AVLNode \*left;

struct AVLNode \*right;

int height;

};

int max(int a, int b) {

return (a > b) ? a : b;

}

int height(struct AVLNode \*node) {

if (node == NULL)

return 0;

return node->height;

}

struct AVLNode \*newNode(int key) {

struct AVLNode \*node = (struct AVLNode \*)malloc(sizeof(struct AVLNode));

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1;

return node;

}

struct AVLNode \*rightRotate(struct AVLNode \*y) {

struct AVLNode \*x = y->left;

struct 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;

}

struct AVLNode \*leftRotate(struct AVLNode \*x) {

struct AVLNode \*y = x->right;

struct 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(struct AVLNode \*node) {

if (node == NULL)

return 0;

return height(node->left) - height(node->right);

}

struct AVLNode \*insert(struct 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 + 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;

}

void inorder(struct AVLNode \*root) {

if (root != NULL) {

inorder(root->left);

printf("%d ", root->key);

inorder(root->right);

}

}

void deleteTree(struct AVLNode \*root) {

if (root != NULL) {

deleteTree(root->left);

deleteTree(root->right);

free(root);

}

}

int main() {

struct AVLNode \*root = NULL;

int choice, key;

do {

printf("1. Insert\n");

printf("2. Print (Inorder Traversal)\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter key to insert: ");

scanf("%d", &key);

root = insert(root, key);

break;

case 2:

printf("AVL Tree (Inorder Traversal): ");

inorder(root);

printf("\n");

break;

case 3:

printf("Exiting...\n");

break;

default:

printf("Invalid choice. Please enter a valid option.\n");

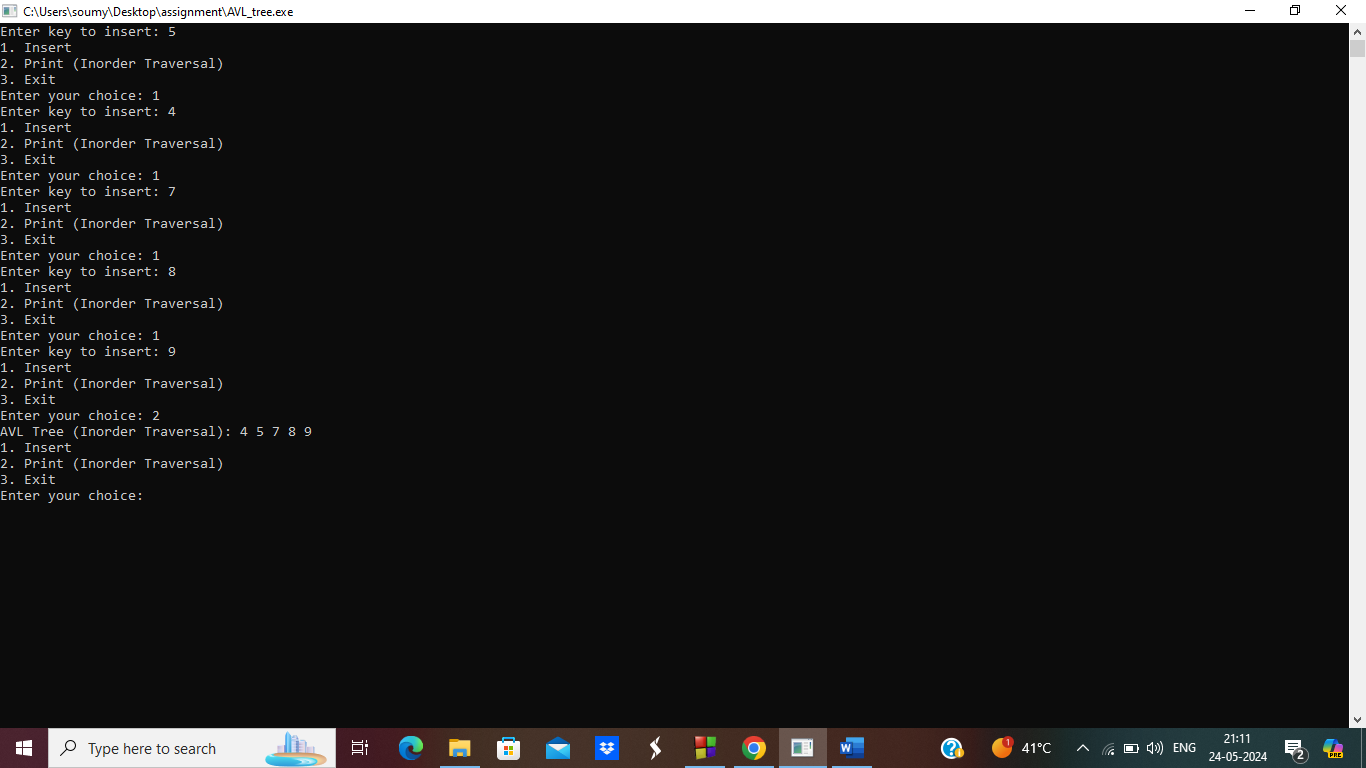
}

} while (choice != 3);

deleteTree(root);

return 0;

}



**2.Print a binary tree.**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

typedef struct Treenode {

int data;

struct Treenode \*left, \*right;

} Treenode;

typedef struct {

Treenode \*root;

} Tree;

Treenode\* newTreenode(int data) {

Treenode\* node = (Treenode\*)malloc(sizeof(Treenode));

node->data = data;

node->left = node->right = NULL;

return node;

}

int height(Treenode \*root) {

if (root == NULL)

return 0;

int left\_height = height(root->left);

int right\_height = height(root->right);

return (left\_height > right\_height ? left\_height : right\_height) + 1;

}

int getcol(int h) {

if (h == 1)

return 1;

return getcol(h - 1) + getcol(h - 1) + 1;

}

void printTree(int \*\*M, Treenode \*root, int col, int row, int height) {

if (root == NULL)

return;

M[row][col] = root->data;

printTree(M, root->left, col - pow(2, height - 2), row + 1, height - 1);

printTree(M, root->right, col + pow(2, height - 2), row + 1, height - 1);

}

void TreePrinter(Tree tree) {

int h = height(tree.root);

int col = getcol(h);

int \*\*M = (int \*\*)malloc(h \* sizeof(int \*));

for (int i = 0; i < h; i++) {

M[i] = (int \*)malloc(col \* sizeof(int));

for (int j = 0; j < col; j++) {

M[i][j] = 0;

}

}

printTree(M, tree.root, col / 2, 0, h);

for (int i = 0; i < h; i++) {

for (int j = 0; j < col; j++) {

if (M[i][j] == 0)

printf(" ");

else

printf("%d ", M[i][j]);

}

printf("\n");

}

for (int i = 0; i < h; i++) {

free(M[i]);

}

free(M);

}

Treenode\* insertLevelOrder(int arr[], Treenode\* root, int i, int n) {

if (i < n) {

Treenode \*temp = newTreenode(arr[i]);

root = temp;

root->left = insertLevelOrder(arr, root->left, 2 \* i + 1, n);

root->right = insertLevelOrder(arr, root->right, 2 \* i + 2, n);

}

return root;

}

int main() {

Tree myTree;

myTree.root = NULL;

int n;

printf("Enter the number of nodes in the tree: ");

scanf("%d", &n);

int \*arr = (int \*)malloc(n \* sizeof(int));

printf("Enter the nodes in level order:\n");

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

myTree.root = insertLevelOrder(arr, myTree.root, 0, n);

printf("Tree structure:\n");

TreePrinter(myTree);

free(arr);

return 0;

}

