**DSC LAB – {9}**

Akshaya agarwal

2247207

3-MCA B

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

typedef struct node {

int key;

int height;

struct node \* left;

struct node \* right;

}

node;

node \* create\_node(int key) {

node \* new\_node = (node \* ) malloc(sizeof(node));

new\_node -> key = key;

new\_node -> height = 1;

new\_node -> left = NULL;

new\_node -> right = NULL;

return new\_node;

}

int get\_height(node \* n) {

if (n == NULL)

return 0;

return n -> height;

}

int get\_balance(node \* n) {

if (n == NULL)

return 0;

return get\_height(n -> left) - get\_height(n -> right);

}

node \* right\_rotate(node \* y) {

node \* x = y -> left;

node \* t2 = x -> right;

x -> right = y;

y -> left = t2;

y -> height = 1 + (get\_height(y -> left) > get\_height(y -> right) ? get\_height(y -> left) : get\_height(y -> right));

x -> height = 1 + (get\_height(x -> left) > get\_height(x -> right) ? get\_height(x -> left) : get\_height(x -> right));

return x;

}

node \* left\_rotate(node \* x) {

node \* y = x -> right;

node \* t2 = y -> left;

y -> left = x;

x -> right = t2;

x -> height = 1 + (get\_height(x -> left) > get\_height(x -> right) ? get\_height(x -> left) : get\_height(x -> right));

y -> height = 1 + (get\_height(y -> left) > get\_height(y -> right) ? get\_height(y -> left) : get\_height(y -> right));

return y;

}

node \* insert(node \* root, int key) {

if (root == NULL)

return create\_node(key);

if (key < root -> key)

root -> left = insert(root -> left, key);

else if (key > root -> key)

root -> right = insert(root -> right, key);

else

return root;

root -> height = 1 + (get\_height(root -> left) > get\_height(root -> right) ? get\_height(root -> left) : get\_height(root -> right));

int balance = get\_balance(root);

if (balance > 1 && key < root -> left -> key)

return right\_rotate(root);

if (balance < -1 && key > root -> right -> key)

return left\_rotate(root);

if (balance > 1 && key > root -> left -> key) {

root -> left = left\_rotate(root -> left);

return right\_rotate(root);

}

if (balance < -1 && key < root -> right -> key) {

root -> right = right\_rotate(root -> right);

return left\_rotate(root);

}

return root;

}

void print\_preorder(node \* root) {

if (root != NULL) {

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

print\_preorder(root -> left);

print\_preorder(root -> right);

}

}

void delete\_tree(node \* root) {

if (root != NULL) {

delete\_tree(root -> left);

delete\_tree(root -> right);

free(root);

}

}

int main() {

node \* root = NULL;

int choice, key;

do {

printf("\nAVL Tree Menu\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("1. Insert Node\n");

printf("2. Print Pre-order\n");

printf("3. Delete Tree\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", & choice);

switch (choice) {

case 1:

printf("Enter key to insert: ");

while (scanf("%d", & key) != 1) {

printf("Invalid input. Enter a numeric value: ");

while (getchar() != '\n');

}

root = insert(root, key);

printf("Node inserted successfully.\n");

break;

case 2:

printf("Pre-order traversal: ");

print\_preorder(root);

printf("\n");

break;

case 3:

delete\_tree(root);

root = NULL;

printf("Tree deleted successfully.\n");

break;

case 4:

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

break;

default:

printf("Invalid choice. Try again.\n");

break;

}

} while (choice != 6);

return 0;

}





