**NAME:** MD. MAHDIR RAHMAN

**ID:** 23-54992-3

**COURSE:** DATA STRUCTURE LAB

**SECTION:** T

**FACULTY NAME:** RAIMA ADHIKARY

**2(a)**

#include <iostream>

#include <stack>

#include <string>

using namespace std;

string infixToPostfix(string infix) {

stack<char> operators;

string postfix = "";

for (char &c : infix) {

if (isalnum(c)) {

postfix += c;

} else if (c == '(') {

operators.push(c);

} else if (c == ')') {

while (!operators.empty() && operators.top() != '(') {

postfix += operators.top();

operators.pop();

}

operators.pop();

} else {

while (!operators.empty() && precedence(operators.top()) >= precedence(c)) {

postfix += operators.top();

operators.pop();

}

operators.push(c);

}

}

while (!operators.empty()) {

postfix += operators.top();

operators.pop();

}

return postfix;

}

int main() {

string infix = "((A+B)-C\*(D/E))+F";

cout << "Postfix: " << infixToPostfix(infix) << endl;

return 0;

}

**2(b)**

#include <iostream>

using namespace std;

struct Node {

int data;

Node \*left, \*right;

Node(int val) : data(val), left(NULL), right(NULL) {}

};

Node\* insert(Node\* root, int val) {

if (!root) return new Node(val);

if (val < root->data) root->left = insert(root->left, val);

else root->right = insert(root->right, val);

return root;

}

void inOrder(Node\* root) {

if (!root) return;

inOrder(root->left);

cout << root->data << " ";

inOrder(root->right);

}

int getHeight(Node\* root) {

if (!root) return 0;

int leftHeight = getHeight(root->left);

int rightHeight = getHeight(root->right);

return max(leftHeight, rightHeight) + 1;

}

int main() {

Node\* root = nullptr;

root = insert(root, 10);

insert(root, 20);

insert(root, 30);

insert(root, 40);

insert(root, 50);

insert(root, 60);

cout << "In-order Traversal: ";

inOrder(root);

cout << endl;

cout << "Height of BST: " << getHeight(root) << endl;

return 0;

}

**3(a)**

#include <iostream>

using namespace std;

struct Node {

char data;

Node \*left, \*right;

Node(char val) : data(val), left(NULL), right(NULL) {}

};

int findIndex(char in[], int start, int end, char value) {

for (int i = start; i <= end; i++) {

if (in[i] == value) {

return i;

}

}

return -1;

}

Node\* buildTree(char pre[], char in[], int &preIdx, int start, int end) {

if (start > end) return NULL;

char curr = pre[preIdx++];

Node\* root = new Node(curr);

if (start == end) return root;

int inIdx = findIndex(in, start, end, curr);

root->left = buildTree(pre, in, preIdx, start, inIdx - 1);

root->right = buildTree(pre, in, preIdx, inIdx + 1, end);

return root;

}

void inOrder(Node\* root) {

if (!root) return;

inOrder(root->left);

cout << root->data << " ";

inOrder(root->right);

}

int main() {

char pre[] = {'K', 'G', 'L', 'M', 'H', 'D', 'B', 'E', 'C', 'A'};

char in[] = {'G', 'D', 'L', 'H', 'M', 'B', 'A', 'E', 'C'};

int preIdx = 0;

Node\* root = buildTree(pre, in, preIdx, 0, 8);

cout << "In-order Traversal: ";

inOrder(root);

cout << endl;

return 0;

}

**3(b)**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(NULL), right(NULL) {}

};

Node\* createBalancedBST(int arr[], int start, int end) {

if (start > end) return NULL;

int mid = (start + end) / 2;

Node\* root = new Node(arr[mid]);

root->left = createBalancedBST(arr, start, mid - 1);

root->right = createBalancedBST(arr, mid + 1, end);

return root;

}

void inOrderTraversal(Node\* root) {

if (!root) return;

inOrderTraversal(root->left);

cout << root->data << " ";

inOrderTraversal(root->right);

}

void printTree(Node\* root, int space = 0, int height = 10) {

if (!root || space > height) return;

space += 5;

printTree(root->right, space);

for (int i = 5; i < space; i++)

cout << " ";

cout << root->data << "\n";

printTree(root->left, space);

}

int main() {

int arr[] = {10, 20, 30, 40, 50, 60};

int n = sizeof(arr) / sizeof(arr[0]);

Node\* root = createBalancedBST(arr, 0, n - 1);

cout << "In-order Traversal of Balanced BST: ";

inOrderTraversal(root);

cout << endl;

cout << "Visual Representation of the Balanced Tree:" << endl;

printTree(root);

return 0;

}