//Assignment 04

#include <iostream>

#include <stack>

#include <string>

#include <cctype>

using namespace std;

struct TreeNode

{

char value;

TreeNode\* left;

TreeNode\* right;

TreeNode(char val) : value(val), left(NULL), right(NULL) {}

};

// Construct Tree from Postfix Expression

TreeNode\* constructTreeFromPostfix(const string& postfix)

{

stack<TreeNode\*> s;

for (size\_t i = 0; i < postfix.length(); ++i)

{

char ch = postfix[i];

if (isdigit(ch) || isalpha(ch))

{

s.push(new TreeNode(ch));

}

else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/')

{

TreeNode\* right = s.top(); s.pop();

TreeNode\* left = s.top(); s.pop();

TreeNode\* node = new TreeNode(ch);

node->left = left;

node->right = right;

s.push(node);

}

}

return s.top();

}

// Construct Tree from Prefix Expression

TreeNode\* constructTreeFromPrefix(const string& prefix)

{

stack<TreeNode\*> s;

for (int i = prefix.length() - 1; i >= 0; --i)

{

char ch = prefix[i];

if (isdigit(ch) || isalpha(ch))

{

s.push(new TreeNode(ch));

}

else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/')

{

TreeNode\* left = s.top(); s.pop();

TreeNode\* right = s.top(); s.pop();

TreeNode\* node = new TreeNode(ch);

node->left = left;

node->right = right;

s.push(node);

}

}

return s.top();

}

// Recursive Traversals

void inorderRecursive(TreeNode\* root)

{

if (root)

{

inorderRecursive(root->left);

cout << root->value << ' ';

inorderRecursive(root->right);

}

}

void preorderRecursive(TreeNode\* root)

{

if (root)

{

cout << root->value << ' ';

preorderRecursive(root->left);

preorderRecursive(root->right);

}

}

void postorderRecursive(TreeNode\* root)

{

if (root)

{

postorderRecursive(root->left);

postorderRecursive(root->right);

cout << root->value << ' ';

}

}

// Non-Recursive Traversals

void inorderNonRecursive(TreeNode\* root)

{

stack<TreeNode\*> s;

TreeNode\* curr = root;

while (curr != NULL || !s.empty())

{

while (curr != NULL)

{

s.push(curr);

curr = curr->left;

}

curr = s.top(); s.pop();

cout << curr->value << ' ';

curr = curr->right;

}

}

void preorderNonRecursive(TreeNode\* root)

{

if (root == NULL) return;

stack<TreeNode\*> s;

s.push(root);

while (!s.empty())

{

TreeNode\* node = s.top(); s.pop();

cout << node->value << ' ';

if (node->right) s.push(node->right);

if (node->left) s.push(node->left);

}

}

void postorderNonRecursive(TreeNode\* root)

{

if (root == NULL) return;

stack<TreeNode\*> s1, s2;

s1.push(root);

while (!s1.empty())

{

TreeNode\* node = s1.top(); s1.pop();

s2.push(node);

if (node->left) s1.push(node->left);

if (node->right) s1.push(node->right);

}

while (!s2.empty()) {

cout << s2.top()->value << ' ';

s2.pop();

}

}

int main()

{

string postfix = "ab+cde+\*\*";

string prefix = "\*+abc/de";

TreeNode\* postfixTree = constructTreeFromPostfix(postfix);

TreeNode\* prefixTree = constructTreeFromPrefix(prefix);

cout << "Postfix Tree Traversals:" << endl;

cout << "In-order: ";

inorderRecursive(postfixTree);

cout << "\nPre-order: ";

preorderRecursive(postfixTree);

cout << "\nPost-order: ";

postorderRecursive(postfixTree);

cout << "\nNon-recursive Traversals:" << endl;

cout << "In-order: ";

inorderNonRecursive(postfixTree);

cout << "\nPre-order: ";

preorderNonRecursive(postfixTree);

cout << "\nPost-order: ";

postorderNonRecursive(postfixTree);

cout << "\n\nPrefix Tree Traversals:" << endl;

cout << "In-order: ";

inorderRecursive(prefixTree);

cout << "\nPre-order: ";

preorderRecursive(prefixTree);

cout << "\nPost-order: ";

postorderRecursive(prefixTree);

cout << "\nNon-recursive Traversals:" << endl;

cout << "In-order: ";

inorderNonRecursive(prefixTree);

cout << "\nPre-order: ";

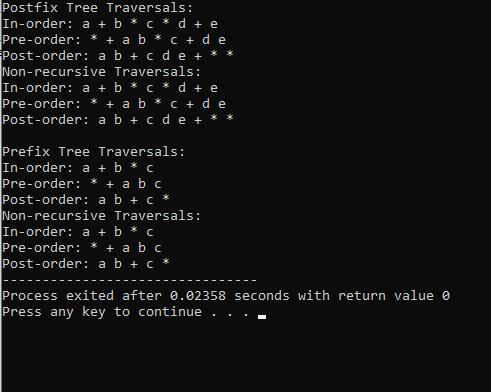
preorderNonRecursive(prefixTree);

cout << "\nPost-order: ";

postorderNonRecursive(prefixTree);

return 0;

}



Git-Hub Link : https://github.com/ArwaSaluji/DSA-Assignment