#include <iostream>

#include <stack>

using namespace std;

// Node structure for the expression tree

struct Node {

char data;

Node\* left;

Node\* right;

Node(char val) {

data = val;

left = nullptr;

right = nullptr;

}

};

// Function to check if a character is an operand

bool isOperand(char c) {

return isalpha(c);

}

// Function to construct an expression tree from prefix expression

Node\* constructExpressionTree(string prefixExpression) {

stack<Node\*> st;

for (int i = prefixExpression.size() - 1; i >= 0; i--) {

char c = prefixExpression[i];

if (isOperand(c)) {

st.push(new Node(c));

} else {

Node\* newNode = new Node(c);

newNode->left = st.top();

st.pop();

newNode->right = st.top();

st.pop();

st.push(newNode);

}

}

return st.top();

}

// Function to perform postorder traversal of the expression tree (non-recursive)

void postorderTraversal(Node\* root) {

if (root == nullptr) return;

stack<Node\*> st;

Node\* prev = nullptr;

st.push(root);

while (!st.empty()) {

Node\* curr = st.top();

if (!prev || prev->left == curr || prev->right == curr) {

if (curr->left) {

st.push(curr->left);

} else if (curr->right) {

st.push(curr->right);

}

} else if (curr->left == prev) {

if (curr->right) {

st.push(curr->right);

}

} else {

cout << curr->data << " ";

st.pop();

}

prev = curr;

}

}

// Function to delete the entire expression tree

void deleteTree(Node\* root) {

if (root == nullptr) return;

deleteTree(root->left);

deleteTree(root->right);

delete root;

}

int main() {

string prefixExpression;

cout << "Enter the prefix expression: ";

cin >> prefixExpression;

Node\* root = constructExpressionTree(prefixExpression);

cout << "Postorder Traversal: ";

postorderTraversal(root);

cout << endl;

// Delete the entire tree to free up memory

deleteTree(root);

return 0;

}