```
/* Expression Tree: Construct an Expression Tree from postfix and prefix expression. Perform pre-order and post-order traversals.*/#include <iostream>
#include <stack>
using namespace std;

// Structure to represent a node in the expression tree
struct TreeNode {
    char data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(char val) : data(val), left(nullptr), right(nullptr) {}
};

// Function to construct an expression tree from a postfix expression
TreeNode* constructExpressionTreeFromPostfix(const string& postfix) {
    stack<TreeNode*> stk;

    for (char ch : postfix) {
        if (isalnum(ch)) {
            stk.push(new TreeNode(ch));
        } else {
            TreeNode* operand2 = stk.top(); stk.pop();
            TreeNode* operand1 = stk.top(); stk.pop();
            TreeNode* newNode = new TreeNode(ch);
            newNode->left = operand1;
            newNode->right = operand2;
            stk.push(newNode);
        }
    }
```

```
    return stk.top();
}

// Pre-order traversal of the expression tree
void preOrderTraversal(TreeNode* root) {
    if (root) {
        cout << root->data << " ";
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}

// Post-order traversal of the expression tree
void postOrderTraversal(TreeNode* root) {
    if (root) {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        cout << root->data << " ";
    }
}

int main() {
    string postfixExpression;
    cout << "Enter postfix expression: ";
    cin >> postfixExpression;

    TreeNode* root = constructExpressionTreeFromPostfix(postfixExpression);
```

```
    cout << "Pre-order traversal: ";
    preOrderTraversal(root);
    cout << endl;

    cout << "Post-order traversal: ";
    postOrderTraversal(root);
    cout << endl;

    return 0;
}
```