```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

// Define TreeNode structure
struct TreeNode {
    char data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(char c) : data(c), left(nullptr), right(nullptr) {}
};

// Check if character is an operator
bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}

// Construct expression tree from prefix expression
TreeNode* constructExpressionTree(const string& prefix) {
    stack<TreeNode*> st;

    for (int i = prefix.length() - 1; i >= 0; i--) {
        char c = prefix[i];

        TreeNode* newNode = new TreeNode(c);

        if (isOperator(c)) {
            if (!st.empty()) {
                newNode->left = st.top();
                st.pop();
            }
            if (!st.empty()) {
```

```
            newNode->right = st.top();
            st.pop();
        }
    }

    st.push(newNode);
  }

  return st.empty() ? nullptr : st.top();
}

// Post-order traversal (non-recursive)
void postOrderTraversal(TreeNode* root) {
  if (root == nullptr) {
    cout << "Tree is empty." << endl;
    return;
  }

  stack<TreeNode*> s1;
  stack<char> s2;

  s1.push(root);

  while (!s1.empty()) {
    TreeNode* node = s1.top();
    s1.pop();

    s2.push(node->data);

    if (node->left)
      s1.push(node->left);
    if (node->right)
      s1.push(node->right);
  }
```

```
    cout << "Post-order traversal: ";
    while (!s2.empty()) {
        cout << s2.top() << " ";
        s2.pop();
    }
    cout << endl;
}

// Delete the entire expression tree
void deleteTree(TreeNode* root) {
    if (root == nullptr) {
        cout << "Tree is already empty." << endl;
        return;
    }

    stack<TreeNode*> s;
    stack<TreeNode*> nodesToDelete;

    s.push(root);

    while (!s.empty()) {
        TreeNode* node = s.top();
        s.pop();

        nodesToDelete.push(node);

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

    while (!nodesToDelete.empty()) {
```

```
        TreeNode* node = nodesToDelete.top();
        nodesToDelete.pop();
        delete node;
    }

    cout << "Tree deleted successfully." << endl;
}

int main() {
    TreeNode* root = nullptr;
    string prefix;
    int choice;

    do {
        cout << "\n----- EXPRESSION TREE MENU -----" << endl;
        cout << "1. Construct Tree from Prefix Expression" << endl;
        cout << "2. Post-order Traversal" << endl;
        cout << "3. Delete Tree" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                // Delete existing tree if any
                if (root != nullptr) {
                    deleteTree(root);
                    root = nullptr;
                }

                cout << "Enter prefix expression: ";
                cin >> prefix;
                root = constructExpressionTree(prefix);
                cout << "Expression tree constructed." << endl;
```

```
            break;

        case 2:
            postOrderTraversal(root);
            break;

        case 3:
            deleteTree(root);
            root = nullptr;
            break;

        case 4:
            cout << "Exiting program..." << endl;
            break;

        default:
            cout << "Invalid choice. Please try again." << endl;
      }
   } while (choice != 4);

   // Clean up before exit
   if (root != nullptr) {
      deleteTree(root);
   }

   return 0;
}
```