```
//Name: 4. Expression Tree
#include <iostream>
 #include <stack>
 #include <string>
 using namespace std;
 struct Node {
   char value;
   Node* le ;
   Node* right;
   Node(char val) : value(val), le (nullptr), right(nullptr) {}
 };
Node* constructPos ixTree(const string& pos ix) {
   stack<Node*> s;
   for (char c : pos ix) {
      if (isalnum(c)) {
         s.push(new Node(c));
      } else {
         Node* operatorNode = new Node(c);
         operatorNode->right = s.top(); s.pop();
         operatorNode->le = s.top(); s.pop();
         s.push(operatorNode);
      }
   }
   return s.top();
```

```
}
Node* constructPrefixTree(const string& prefix) {
    stack<Node*> s;
    for (int i = prefix.length() - 1; i >= 0; i--) {
        char c = prefix[i];
        if (isalnum(c)) {
            s.push(new Node(c));
        } else {
            Node* operatorNode = new Node(c);
            operatorNode->le = s.top(); s.pop();
            operatorNode->right = s.top(); s.pop();
            s.push(operatorNode);
        }
    }
    return s.top();
 }
void inOrder(Node* root) {
    if (!root) return;
    inOrder(root->le );
    cout << root->value << " ";
    inOrder(root->right);
 }
 void preOrder(Node* root) {
    if (!root) return;
```

```
    cout << root->value << " ";
    preOrder(root->le );
    preOrder(root->right);
 }
void postOrder(Node* root) {
    if (!root) return;
    postOrder(root->le );
    postOrder(root->right);
    cout << root->value << " ";
 }
void nonRecursiveInOrder(Node* root) {
    stack<Node*> s;
    Node* current = root;
    while (current || !s.empty()) {
        while (current) {
            s.push(current);
            current = current->le ;
        }
        current = s.top(); s.pop();
        cout << current->value << " ";
        current = current->right;
    }
 }
void nonRecursivePreOrder(Node* root) {
```

```
  if (!root) return;
  stack<Node*> s;
  s.push(root);
  while (!s.empty()) {
     Node* current = s.top(); s.pop();
     cout << current->value << " ";
     if (current->right) s.push(current->right);
     if (current->le ) s.push(current->le );
  }
}
// Non-recursive post-order traversal
void nonRecursivePostOrder(Node* root) {
  if (!root) return;
  stack<Node*> s1, s2;
  s1.push(root);
  while (!s1.empty()) {
     Node* current = s1.top(); s1.pop();
     s2.push(current);
     if (current->le ) s1.push(current->le );
     if (current->right) s1.push(current->right);
  }
  while (!s2.empty()) {
     cout << s2.top()->value << " ";
     s2.pop();
```

```
    }
 }
int main() {
    string pos ix = "ab+c*";
    string prefix = "*+abc";
    Node* pos ixRoot = constructPos ixTree(pos ix);
    Node* prefixRoot = constructPrefixTree(prefix);
    cout << "Pos ix Expression Tree Traversals (ab+c*):\n";
    cout << "In-order (Recursive): "; inOrder(pos ixRoot); cout << endl;
    cout << "Pre-order (Recursive): "; preOrder(pos ixRoot); cout <<
endl;
    cout << "Post-order (Recursive): "; postOrder(pos ixRoot); cout <<
endl;
    cout << "In-order (Non-Recursive): ";
nonRecursiveInOrder(pos ixRoot); cout <<
endl;
cout << "Pre-order (Non-Recursive): ";
nonRecursivePreOrder(pos ixRoot); cout
<< endl;
cout << "Post-order (Non-Recursive): ";
nonRecursivePostOrder(pos ixRoot); cout
<< endl;
cout << "\nPrefix Expression Tree Traversals (*+abc):\n";
cout << "In-order (Recursive): "; inOrder(prefixRoot); cout << endl;
```

```
cout << "Pre-order (Recursive): "; preOrder(prefixRoot); cout << endl;
cout << "Post-order (Recursive): "; postOrder(prefixRoot); cout <<
endl;
cout << "In-order (Non-Recursive): "; nonRecursiveInOrder(prefixRoot);
cout <<
endl;
cout << "Pre-order (Non-Recursive): ";
nonRecursivePreOrder(prefixRoot); cout <<
endl;
cout << "Post-order (Non-Recursive): ";
nonRecursivePostOrder(prefixRoot); cout
<< endl;
return 0;
}
```