```
#include <iostream>
#include <stack>
using namespace std;
struct Node
{
 char data;
 Node* left;
 Node* right;
};
int isOperator(char c)
{
 return (c == '+' || c == '-' || c == '*' || c == '/');
}
Node* createNode(char d)
{
 Node* newNode = new Node;
 newNode->data = d;
 newNode->left = NULL;
 newNode->right = NULL;
 return newNode;
}
Node* constructExpressionTree(const string& postfix)
 {
 stack<Node*> st;
 for (char c : postfix)
 {
 if (!isOperator(c))
 st.push(createNode(c));
 else
 {
 Node* rightOperand = st.top(); st.pop();
 Node* leftOperand = st.top(); st.pop();
 Node* newNode = createNode(c);
newNode->left = leftOperand;

 newNode->right = rightOperand;
 st.push(newNode);

 }

 }

 Return st.top();

 }

Void inorderTraversal(Node* root)

{

 If (root)

 {

 inorderTraversal(root->left);
 cout << root->data << " ";
 inorderTraversal(root->right);

 }

 }
```

```
Void preorderTraversal(Node* root)

{

 If (root)

 {

 Cout << root->data << " ";

 preorderTraversal(root->left);

 preorderTraversal(root->right);

 }

}

Void postorderTraversal(Node* root)

 {

 If (root)

 {

 postorderTraversal(root->left);

 postorderTraversal(root->right);

 cout << root->data << " ";
 }

}

Int main()

 {

 String postfix;
Cout << “Enter Postfix Expression: “;

 Cin >> postfix;

 Node* root = constructExpressionTree(postfix);

 Cout << “In-Order Traversal: “;

 inorderTraversal(root);

 cout << “\nPre-Order Traversal: “;

 preorderTraversal(root);

 cout << “\nPost-Order Traversal: “;

 postorderTraversal(root);

 return 0;

}
```

OUTPUT:

```
Enter Postfix Expression: xy*z+

In-Order Traversal: x * y + z

Pre-Order Traversal: + * x y z

Post-Order Traversal: x y * z +
```