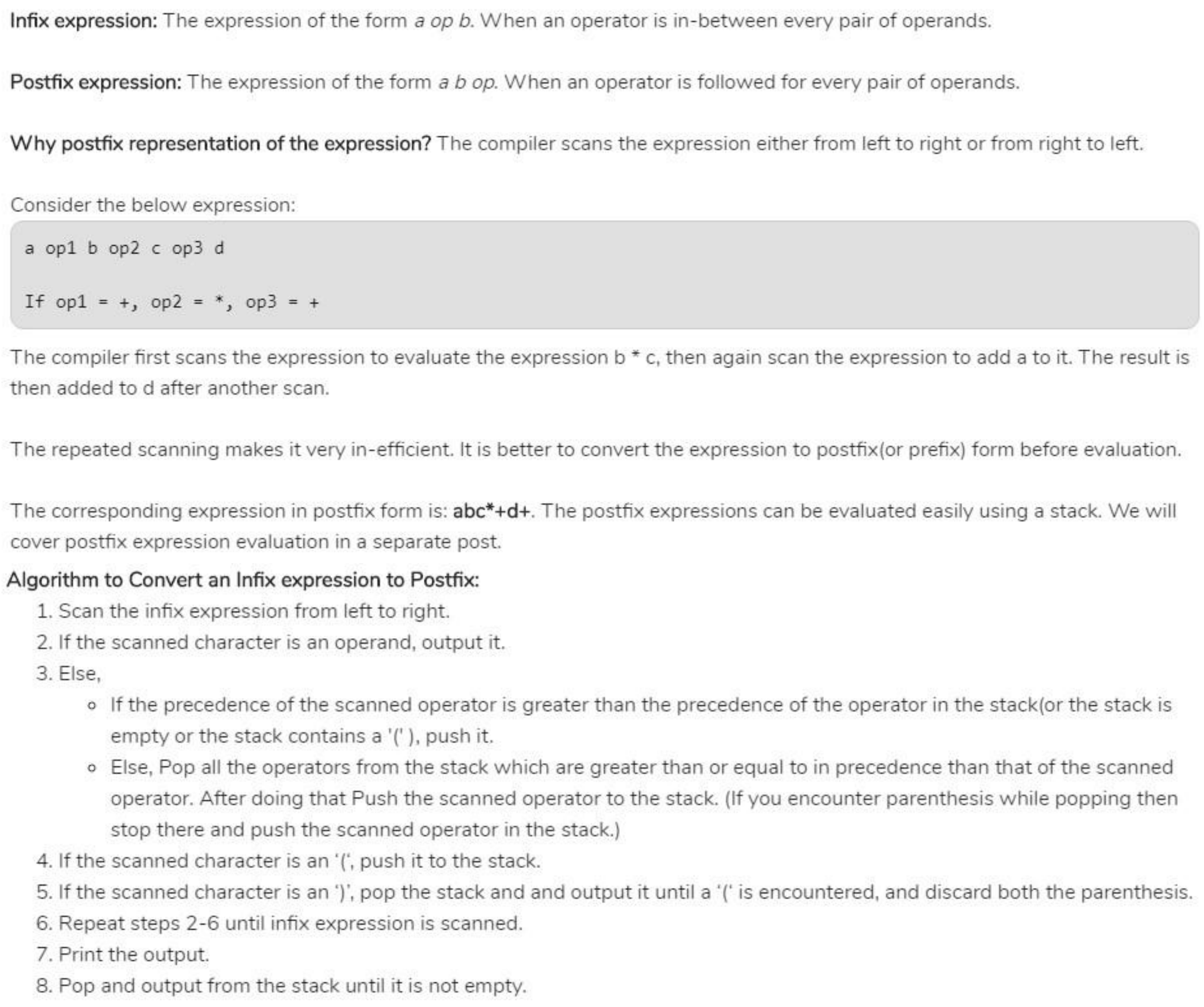
Infix to postfix using stacks



Infix expression: The expression of the form *a op b*. When an operator is in-between every pair of operands.

Postfix expression: The expression of the form *a b op*. When an operator is followed for every pair of operands.

Why postfix representation of the expression? The compiler scans the expression either from left to right or from right to left.

Consider the below expression:

```
a op1 b op2 c op3 d

If op1 = +, op2 = *, op3 = +
```

The compiler first scans the expression to evaluate the expression b * c, then again scan the expression to add a to it. The result is then added to d after another scan.

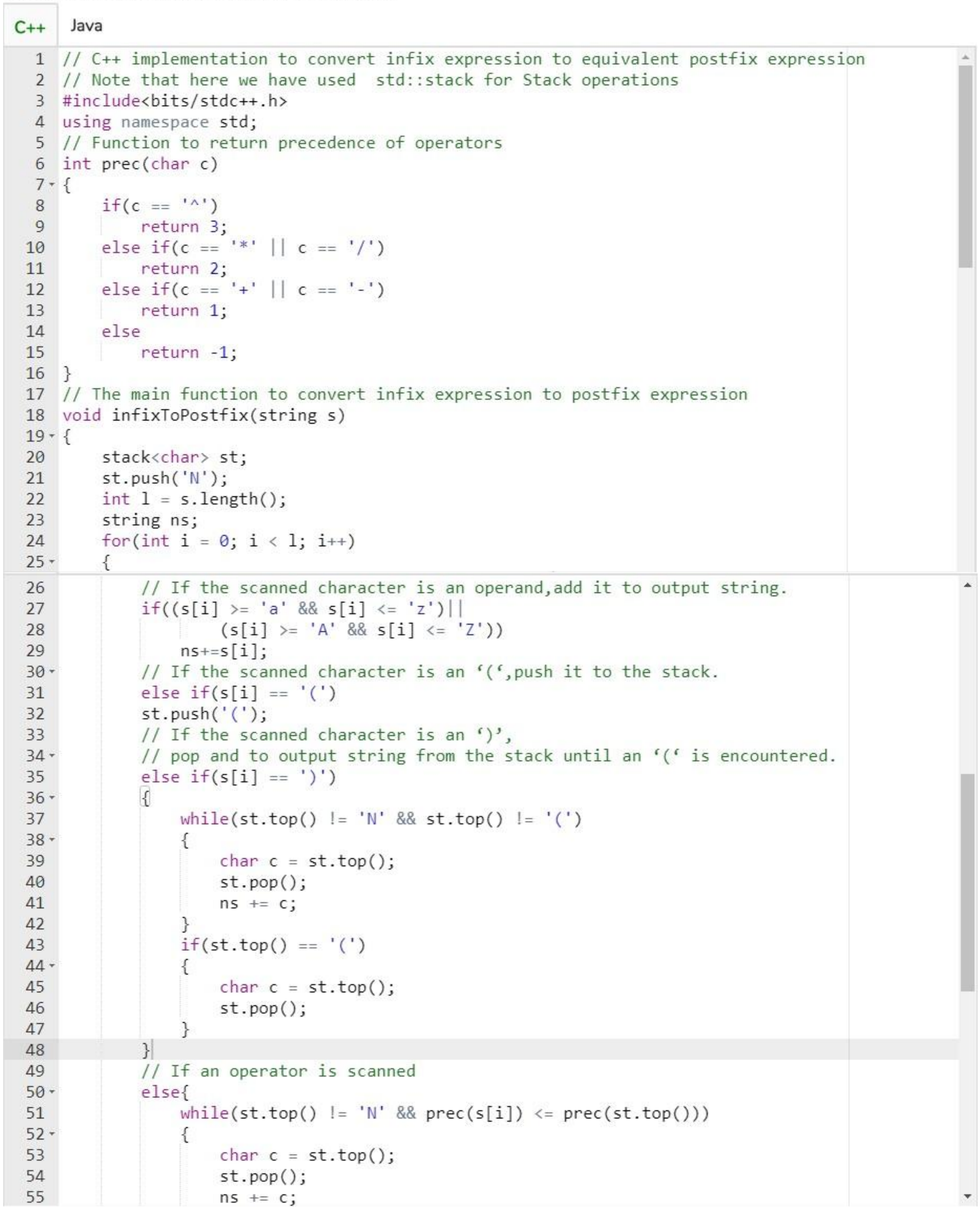
The repeated scanning makes it very in-efficient. It is better to convert the expression to postfix(or prefix) form before evaluation.

The corresponding expression in postfix form is: **abc*+d+**. The postfix expressions can be evaluated easily using a stack. We will cover postfix expression evaluation in a separate post.

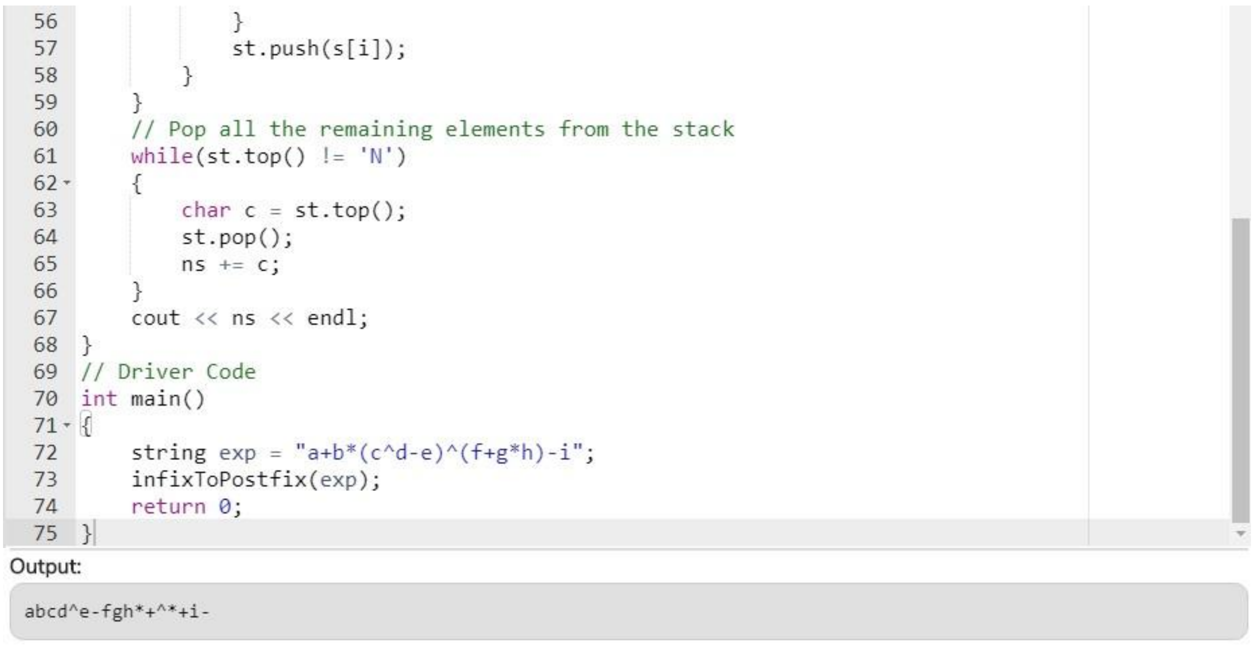
Algorithm to Convert an Infix expression to Postfix:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else,
    - If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.
    - Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
4. If the scanned character is an '(', push it to the stack.
5. If the scanned character is an ')', pop the stack and and output it until a '(' is encountered, and discard both the parenthesis.
6. Repeat steps 2-6 until infix expression is scanned.
7. Print the output.
8. Pop and output from the stack until it is not empty.

Below is the implementation of the above algorithm:



```cpp
// C++ implementation to convert infix expression to equivalent postfix expression
// Note that here we have used  std::stack for Stack operations
#include<bits/stdc++.h>
using namespace std;
// Function to return precedence of operators
int prec(char c)
{
    if(c == '^')
        return 3;
    else if(c == '*' || c == '/')
        return 2;
    else if(c == '+' || c == '-')
        return 1;
    else
        return -1;
}
// The main function to convert infix expression to postfix expression
void infixToPostfix(string s)
{
    stack<char> st;
    st.push('N');
    int l = s.length();
    string ns;
    for(int i = 0; i < l; i++)
    {
        // If the scanned character is an operand,add it to output string.
        if((s[i] >= 'a' && s[i] <= 'z')||
                (s[i] >= 'A' && s[i] <= 'Z'))
            ns+=s[i];
        // If the scanned character is an '(',push it to the stack.
        else if(s[i] == '(')
        st.push('(');
        // If the scanned character is an ')',
        // pop and to output string from the stack until an '(' is encountered.
        else if(s[i] == ')')
        {
            while(st.top() != 'N' && st.top() != '(')
            {
                char c = st.top();
                st.pop();
                ns += c;
            }
            if(st.top() == '(')
            {
                char c = st.top();
                st.pop();
            }
        }
        // If an operator is scanned
        else{
            while(st.top() != 'N' && prec(s[i]) <= prec(st.top()))
            {
                char c = st.top();
                st.pop();
                ns += c;
```



```cpp
            }
            st.push(s[i]);
        }
    }
    // Pop all the remaining elements from the stack
    while(st.top() != 'N')
    {
        char c = st.top();
        st.pop();
        ns += c;
    }
    cout << ns << endl;
}
// Driver Code
int main()
{
    string exp = "a+b*(c^d-e)^(f+g*h)-i";
    infixToPostfix(exp);
    return 0;
}
```

Output:

```
abcd^e-fgh*+^*+i-
```