Postfix se Infix Conversion using Stack

Introduction

Postfix notation (Reverse Polish Notation) ek aisa mathematical notation hai jisme operator sabhi operands ke baad aata hai. Postfix expression ko infix me convert karne ke liye ek stack ka use kiya jata hai jo order of operations ko maintain karta hai. Iss document me C++ code ka logic, dry run, aur step-by-step explanation di gayi hai.

Thought Process

1. **Stack ka Use:**
   - Ek stack use hota hai jisme operands ko store kiya jata hai.
   - Agar ek operator milta hai, toh top do operands pop hote hain aur ek infix expression form karke stack me wapas push kiya jata hai.

2. **Postfix Expression Traverse Karna:**
   - Agar ek digit (operand) mile, toh usko stack me push karo.
   - Agar ek operator mile:
     - Stack se top ke do elements pop karo.
     - Unhe operator ke saath ek infix expression me convert karo.
     - Naya expression wapas stack me push karo.

3. **Final Result:**
   - Pura postfix expression process hone ke baad stack me sirf ek element bachega jo final infix expression hoga.

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

// Function jo infix string return karega
string eval(string v1, string v2, char ch) {
    string s = "";
    s += v1;
    s.push_back(ch);
    s += v2;
    return s;
}

int main() {
    string s = "126+4*8/+3-"; // Postfix expression
    stack<string> val; // Stack jo operands aur expressions store karega

    for(int i = 0; i < s.length(); i++) {
        if(s[i] >= '0' && s[i] <= '9') { // Agar operand mile toh push karo
            val.push(to_string(s[i] - '0'));
        } else { // Agar operator mile toh do operands pop karo
            string v2 = val.top(); val.pop();
            char ch = s[i];
            string v1 = val.top(); val.pop();
            string ans = eval(v1, v2, ch); // Infix expression banao
            val.push(ans); // Stack me push karo
        }
    }

    // Final infix expression print karo
    cout << "Infix Expression: " << val.top() << endl;
    return 0;
}
```

Dry Run

Example 1
Input: `126+4*8/+3-`

Step	Stack Content	Operation
1	1	Push 1
2	1, 2	Push 2
3	1, 2, 6	Push 6
4	1, 8	6 + 2 = 8
5	1, 8, 4	Push 4
6	1, 8, 16	4 * 4 = 16
7	1, 24	8 + 16 = 24
8	3	Push 3
9	21	24 - 3 = 21

Output: `((1+(2+6))+(4*8))/3-`

Conclusion

- Yeh approach postfix se infix conversion ko stack ka use karke efficiently handle karta hai.
- `eval` function important role play karta hai infix expression construct karne me.
- Final result ek single element ke form me stack me milta hai jo complete infix expression hota hai.

Yeh document detailed explanation, logic, aur dry runs ke saath diya gaya hai jo conversion process ko samajhne me madad karega.