Infix to Postfix Conversion - Detailed Breakdown

Introduction

This document provides a detailed breakdown of the C++ code that converts an infix expression to a postfix expression. The thought process behind writing the code, its structure, working, dry run, and test cases are explained thoroughly. Additionally, a short notes section is provided for quick revision.

Code Snippet

```
#include<iostream>
#include<string>
#include<stack>
using namespace std;
int priority(char ch){
 if(ch=='+'||ch=='-') return 1;
 else if(ch=='*'||ch=='/')return 2;
}
string eval(string v1,string v2,char ch){
 string s = "";
 s += v1;
 s += v2;
 s.push_back(ch);
 return s;
}
int main(){
 string s = "1+(2+6)*4/8-3";
 stack<string> val;
 stack<char> op;
 for(int i=0;i<s.length();i++){
  if(s[i]>=48 && s[i]<=57){
   val.push(to_string(s[i]-48));
  }
  else{
   if(op.size()==0){
    op.push(s[i]);
   }
   else if(s[i]=='('||op.top()=='(') op.push(s[i]);
   else if(op.size()>0 && s[i]==')'){
    while(op.top()!='('){
     string v2 = val.top();
     val.pop();
     string v1 = val.top();
     val.pop();
     char ch = op.top();
     op.pop();
     string ans = eval(v1,v2,ch);
     val.push(ans);
    }
    op.pop();
```

```
    }
    else if(priority(op.top())<priority(s[i])) op.push(s[i]);
    else{
      while(op.size()>0 && priority(op.top())>=priority(s[i])){
      string v2 = val.top();
      val.pop();
      char ch = op.top();
      op.pop();
      string v1 = val.top();
      val.pop();
      string ans = eval(v1,v2,ch);
      val.push(ans);
      }
      op.push(s[i]);
    }
   }
  }
  while(op.size()>0){
      string v2 = val.top();
      val.pop();
      char ch = op.top();
      op.pop();
      string v1 = val.top();
      val.pop();
      string ans = eval(v1,v2,ch);
      val.push(ans);
  }
  cout<<val.top();
  return 0;
}
```

Thought Process Behind the Code

1. **Handling Operands:**
   - If the character is a digit, convert it to a string and push it onto the val stack.
2. **Handling Operators:**
   - If the operator stack is empty, push the operator.
   - If an opening bracket (is found, push it onto the operator stack.
   - If a closing bracket), process operators until an opening bracket is found.
   - If the current operator has higher precedence than the top of the stack, push it.
   - Otherwise, evaluate the top operators before pushing the current operator.
3. **Final Processing:**
   - After the loop, process the remaining operators until the operator stack is empty.

Dry Run (Step-by-Step Execution)

For the input **1+(2+6)*4/8-3**, the postfix expression should be:
126+4*8/+3-

Stepwise Execution

Step	Character	Action
1	1	Push to val (1)
2	+	Push to op (+)

3	(	Push to op (()
4	2	Push to val (2)
5	+	Push to op (+)
6	6	Push to val (6)
7	)	Evaluate (2 6 + → 26+), pop (
8	*	Push to op (*)
9	4	Push to val (4)
10	/	Push to op (/)
11	8	Push to val (8)
12	Evaluate	(4 8 / → 48/), push result
13	Evaluate	(26+ 48/ * → 126+48/*)
14	-	Push to op (-)
15	3	Push to val (3)
16	Evaluate	(126+48/* 3 - → 126+48/+3-)

Final Output

126+4*8/+3-

Short Notes (Quick Revision)

- **Operands (digits)** → Directly push onto val stack.
- **Operators (+, -, , /)* → Use precedence to decide push or evaluate.
- **Parentheses** → Process until (is found.
- **Final Evaluation** → Process remaining operators.
- **Final val.top()** → Contains the postfix expression.

Summary

This C++ program correctly converts an infix expression to a postfix expression using **two stacks** (one for operands and one for operators). The algorithm follows operator precedence rules and correctly handles parentheses. The dry run verifies the correctness of the conversion.

From <https://chatgpt.com/c/67c9d703-c788-8009-a0d9-0c07c5db45b5>