Expression Evaluation using Stacks - Detailed Explanation & Corrected Dry R

This document explains the working of the given C++ code that evaluates an arithmetic expression using stacks. It includes a step-by-step breakdown and corrected dry run examples.

1. priority() Function

The priority() function assigns precedence to operators. '+' and '-' have lower precedence (1), while '*' and '/' have higher precedence (2).

```
int priority(char ch){
  if(ch=='+'||ch=='-') return 1;
  else if(ch=='*'||ch=='/') return 2;
}
```

2. eval() Function

The eval() function takes two operands and an operator, then performs the correct arithmetic operation.

```
int eval(int v1, int v2, char ch){
  if(ch=='+') return v1+v2;
  else if(ch == '-') return v1-v2;
  else if(ch == '*') return v1*v2;
  else return v1/v2;
}
```

3. Main Function Breakdown

The main() function processes the given infix expression using two stacks: one for values (operands) and one for operators. It follows these steps:

1. If the character is a number, push it into the value stack.
2. If it's an operator, check precedence and process accordingly.
3. If '(' is encountered, push it to the operator stack.
4. If ')' is encountered, evaluate until '(' is found.
5. After scanning, process remaining operators in the stack.
6. The final result is stored at the top of the value stack.

4. Corrected Dry Run Example 1: Expression (3+5*2)-8/4

Step-by-step execution with '(' handling:
1. Read '(' -> Push to operator stack [(]
2. Read '3' -> Push to value stack [3]
3. Read '+' -> Push to operator stack [(+]
4. Read '5' -> Push to value stack [3, 5]
5. Read '*' -> Push to operator stack [(+ *]
6. Read '2' -> Push to value stack [3, 5, 2]
7. Read ')' -> Compute 5 * 2 -> Push result [3, 10]
8. Compute 3 + 10 -> Push result [13]
9. Read '-' -> Push to operator stack [-]
10. Read '8' -> Push to value stack [13, 8]
11. Read '/' -> Push to operator stack [- /]
12. Read '4' -> Push to value stack [13, 8, 4]
13. Compute 8 / 4 -> Push result [13, 2]
14. Compute 13 - 2 -> Final result: 11

5. Corrected Dry Run Example 2: Expression 6/(2+1)*4

Step-by-step execution:
1. Read '6' -> Push to value stack [6]
2. Read '/' -> Push to operator stack [/]
3. Read '(' -> Push to operator stack [(]
4. Read '2' -> Push to value stack [6, 2]
5. Read '+' -> Push to operator stack [(+]
6. Read '1' -> Push to value stack [6, 2, 1]
7. Read ')' -> Compute 2 + 1 -> Push result [6, 3]
8. Compute 6 / 3 -> Push result [2]
9. Read '*' -> Push to operator stack [*]
10. Read '4' -> Push to value stack [2, 4]
11. Compute 2 * 4 -> Final result: 8