# Expression Evaluation using Stacks - Hinglish Explanation

Yeh document C++ code ka explanation deta hai jo ek arithmetic expression ko stacks ka use karke evaluate karta hai. Yahaan ek ek step ka breakdown aur dry run example diye gaye hain.

## 1. priority() Function

priority() function har operator ko precedence assign karta hai. '+' aur '-' ki priority low (1) hai, jabki '*' aur '/' ki priority high (2) hai.

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

## 2. eval() Function

eval() function do operands aur ek operator leta hai, aur uske according arithmetic operation perform karta hai.

```
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

main() function given infix expression ko do stacks ka use karke process karta hai:
1. Agar character number hai, to value stack me push hota hai.
2. Agar operator hai, to precedence check karke process hota hai.
3. Agar '(' aata hai, to operator stack me push hota hai.
4. Agar ')' aata hai, to '(' tak evaluate hota hai.
5. Expression scan hone ke baad jo operators bache hain unko evaluate karna hota hai.
6. Final result value stack ke top pe hota hai.

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

Step-by-step execution with '(' handling:

1. '(' read -> Operator stack: [(]
2. '3' read -> Value stack: [3]
3. '+' read -> Operator stack: [(+]
4. '5' read -> Value stack: [3, 5]
5. '*' read -> Operator stack: [(+ *]
6. '2' read -> Value stack: [3, 5, 2]
7. ')' read -> Compute 5 * 2 -> Result: [3, 10]
8. Compute 3 + 10 -> Result: [13]
9. '-' read -> Operator stack: [-]
10. '8' read -> Value stack: [13, 8]
11. '/' read -> Operator stack: [- /]
12. '4' read -> Value stack: [13, 8, 4]
13. Compute 8 / 4 -> Result: [13, 2]
14. Compute 13 - 2 -> Final result: 11

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

Step-by-step execution:

1. '6' read -> Value stack: [6]
2. '/' read -> Operator stack: [/]
3. '(' read -> Operator stack: [/(]
4. '2' read -> Value stack: [6, 2]
5. '+' read -> Operator stack: [/(+]
6. '1' read -> Value stack: [6, 2, 1]
7. ')' read -> Compute 2 + 1 -> Result: [6, 3]
8. Compute 6 / 3 -> Result: [2]
9. '*' read -> Operator stack: [*]
10. '4' read -> Value stack: [2, 4]
11. Compute 2 * 4 -> Final result: 8