

Question 1:

<https://leetcode.com/problems/evaluate-reverse-polish-notation/>

You are given an array of strings tokens that represents an arithmetic expression in a [Reverse Polish Notation](#).

Evaluate the expression. Return an integer that represents the value of the expression.

Note that:

- The valid operators are '+', '-', '*', and '/'.
- Each operand may be an integer or another expression.
- The division between two integers always truncates toward zero.
- There will not be any division by zero.
- The input represents a valid arithmetic expression in a reverse polish notation.
- The answer and all the intermediate calculations can be represented in a 32-bit integer.

My approach:

Make a stack

For every element in the vector

If it is a operand push it in the stack

If it is a operator:

Take out the first 2 operands.

Evaluate the result and put the result back on the stack.

Final result will be the stack top.

The screenshot shows the LeetCode interface for problem 224, "Basic Calculator". The problem is marked as "Accepted". The user's solution is in C++ and has a runtime of 15 ms, beating 46.16% of other solutions, and a memory usage of 12.9 MB, beating 9.93% of other solutions. The code defines a class Solution with a method get_value that takes a string s and returns an integer. The code is as follows:

```
class Solution {
public:
    int get_value(string s)
    {
```

Code:-

```
class Solution {
public:
    int get_value(string s)
    {
```

```

    if(s=="+")
        return 1;
    if(s=="-")
        return 2;
    if(s=="*")
        return 3;
    if(s=="/")
        return 4;
    return -1;
}
int evalRPN(vector<string>& tokens) {
    stack<string> s;
    for(int i =0;i<tokens.size();i++)
    {
        if(tokens.at(i) == "-" || tokens.at(i)=="+" || tokens.at(i)=="*" || tokens.at(i)=="/")
        {
            int value = get_value(tokens.at(i));
            int op1 = stoi(s.top());
            s.pop();
            int op2 = stoi(s.top());
            s.pop();
            int eval;
            switch(value)
            {
                case 1:
                {
                    eval = op2 + op1;
                    break;
                }
                case 2:
                {
                    eval = op2 - op1;
                    break;
                }
                case 3:
                {
                    eval = op2 * op1;
                    break;
                }
                case 4:
                {
                    eval = op2 / op1;
                    break;
                }
            }

```

```
        }
        s.push(to_string(eval));
    }
    else
    {
        s.push(tokens.at(i));
    }
}
return stoi(s.top());
}
};
```