Microsoft:

#1 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.

Example 1:

```
Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

Example 2:

```
Input: tokens = ["4","13","5","/","+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
```

Example 3:

```
Input: tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
Output: 22
Explanation: ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
```

```
int evalRPN(vector<string>& tokens) {
    stack<long long int>s;
    for(auto x:tokens)
    {
      if(x=="+"|| x=="-"||x=="*"||x=="/")
      {
        long long int operator1,operator2;
        operator1=s.top();
        s.pop();
        operator2=s.top();
        s.pop();
        if(x=="+")
          s.push(operator2+operator1);
        if(x=="*")
          s.push(operator2*operator1);
        if(x=="-")
          s.push(operator2-operator1);
        if(x=="/")
          s.push(operator2/operator1);
      }
      else
        s.push(stoll(x));
    }
    return s.top();
  }
```

Microsoft:
#1 Explanation:
Using Stack:

The reverse polish is a mathematical notation in which operators follow their operands. So, we will get the operands first and then the operators.

- We store all the operands in the order we receive it in.
- If we get an operator, we operate it on the previous two operands.
- We store the resultant operand as it will be used for future operations.
- We use a stack to store all the operands.

So the algorithm is:

- *If the character is a number (operand), push it into the stack*
- *If the character is an operator,*
  - *Pop the top two operands (numbers) from the stack.*
  - *Find the result of the operation using the operator*
  - *Push the result back in the stack*
- After traversal, the top of the stack will be the result of evaluated reverse polish expression.

Complexity

- Time complexity: `O(N)`, where `N` is the number of tokens
- Space complexity: `O(N)`, for maintaining the stack

```
int evalRPN(vector<string>& tokens) {
        stack<long long int>s;
        for(auto x:tokens)
        {
            if(x=="+"|| x=="-"||x=="*"||x=="/")
            {
                long long int operator1,operator2;
                operator1=s.top();
                s.pop();
                operator2=s.top();
                s.pop();
                if(x=="+")
                    s.push(operator2+operator1);
                if(x=="*")
                    s.push(operator2*operator1);
                if(x=="-")
                    s.push(operator2-operator1);
                if(x=="/")
                    s.push(operator2/operator1);
            }
            else
                s.push(stoll(x));
        }
        return s.top();
    }
```