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

Constraints:

- 1 <= tokens.length <= 10^4
- tokens[i] is either an operator: "+", "-", "*", or "/", or an integer in the range [-200, 200].

Code of given question :-

```
#include <vector>
#include <stack>
#include <string>

class Solution {
public:
    int evalRPN(vector<string>& tokens) {
        // Create a stack to keep track of integers for evaluation
        stack<int> numbers;

        // Iterate over each token in the Reverse Polish Notation expression
        for (const string& token : tokens) {
            // If the token represents a number (can be multiple digits or negative)
            if (token.size() > 1 || isdigit(token[0])) {
                // Convert the string token to an integer and push onto the stack
                numbers.push(stoi(token));
            } else { // If the token is an operator
                // Pop the second operand from the stack
                int operand2 = numbers.top();
                numbers.pop();
```

```
            // Pop the first operand from the stack
            int operand1 = numbers.top();
            numbers.pop();

            // Perform the operation based on the type of operator
            switch (token[0]) {
                case '+': // Addition
                    numbers.push(operand1 + operand2);
                    break;
                case '-': // Subtraction
                    numbers.push(operand1 - operand2);
                    break;
                case '*': // Multiplication
                    numbers.push(operand1 * operand2);
                    break;
                case '/': // Division
                    numbers.push(operand1 / operand2);
                    break;
            }
        }
    }

    // The final result is the only number remaining on the stack
    return numbers.top();
  }
};
```
