1. Evaluate Reverse Polish Notation

Evaluate the value of an arithmetic expression in [Reverse Polish Notation](http://en.wikipedia.org/wiki/Reverse_Polish_notation).

Valid operators are +, -, \*, /. Each operand may be an integer or another expression.

**Note:**

* Division between two integers should truncate toward zero.
* The given RPN expression is always valid. That means the expression would always evaluate to a result and there won’t be any divide by zero operation.

**Example 1:**

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

**Example 2:**

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

**Example 3:**

Input: ["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

**解** 逆波兰表达式中，两个操作数紧跟一个运算符

class Solution {  
public:  
 int evalRPN(vector<string>& tokens) {  
 stack<int>s1;  
 for(int i = 0; i < tokens.size(); ++i){  
 if(tokens[i].size() == 1 && !isdigit(tokens[i][0])){  
 int x1 = s1.top();  
 s1.pop();  
 int x2 = s1.top();  
 s1.pop();  
 s1.push(compute(x2, x1, tokens[i]));  
 }else{  
 s1.push(stoi(tokens[i]));  
 }  
 }  
 return s1.top();  
 }  
 int compute(int &x1, int &x2, string &op){  
 if(op == "+"){  
 return x1 + x2;  
 }else if(op == "-"){  
 return x1 - x2;  
 }else if(op == "\*"){  
 return x1 \* x2;  
 }else if(op == "/"){  
 return x1 / x2;  
 }  
 return -1;  
 }  
};

Note : 影响代码速度

1. for循环中直接枚举会慢一些
2. 函数传参时，传值会慢一些，尽量使用传引用