# 题目

请你实现一个「数字乘积类」ProductOfNumbers，要求支持下述两种方法：

1、add(int num)

将数字 num 添加到当前数字列表的最后面。

2、getProduct(int k)

返回当前数字列表中，最后 k 个数字的乘积。

你可以假设当前列表中始终至少包含k个数字。

题目数据保证：任何时候，任一连续数字序列的乘积都在 32-bit 整数范围内，不会溢出。

示例：

输入：

["ProductOfNumbers","add","add","add","add","add","getProduct","getProduct","getProduct","add","getProduct"]

[[],[3],[0],[2],[5],[4],[2],[3],[4],[8],[2]]

输出：

[null,null,null,null,null,null,20,40,0,null,32]

解释：

ProductOfNumbers productOfNumbers = new ProductOfNumbers();

productOfNumbers.add(3); // [3]

productOfNumbers.add(0); // [3,0]

productOfNumbers.add(2); // [3,0,2]

productOfNumbers.add(5); // [3,0,2,5]

productOfNumbers.add(4); // [3,0,2,5,4]

productOfNumbers.getProduct(2); // 返回 20 。最后 2 个数字的乘积是 5 \* 4 = 20

productOfNumbers.getProduct(3); // 返回 40 。最后 3 个数字的乘积是 2 \* 5 \* 4 = 40

productOfNumbers.getProduct(4); // 返回 0 。最后 4 个数字的乘积是 0 \* 2 \* 5 \* 4 = 0

productOfNumbers.add(8); // [3,0,2,5,4,8]

productOfNumbers.getProduct(2); // 返回 32 。最后 2 个数字的乘积是 4 \* 8 = 32

提示：

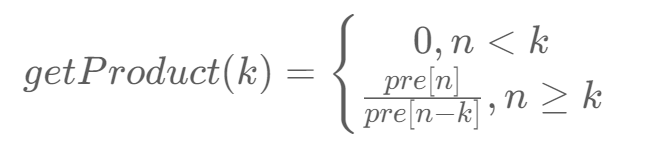
add 和 getProduct 两种操作加起来总共不会超过 40000 次。

0 <= num <= 100

1 <= k <= 40000

# 分析

如果没有0的话，直接考虑维护一个前缀积pre[i]表示前i个数的乘积即可，答案就是pre[n]/pre[n−k]，其中n表示当前pre数组的长度。那么如何处理0呢？可以注意到如果出现0的话，那么0之前的数对答案都是没有用的了，所以我们可以遇到0的时候直接清空pre数组，那么询问的时候我们要求的是末尾k个数的乘积，如果这时候我们pre数组的长度小于k，说明末尾k个数里肯定有0，直接输出0即可，否则输出pre[n]/pre[n−k]，简言之：



其中n=pre.length()  
 代码：

class ProductOfNumbers {

public:

ProductOfNumbers() {

pre[0] = 1;

len = 0;

}

void add(int num) {

// 处理新增为0的情况

if (num == 0) {

len = 0;

} else {

len++;

pre[len] = num;

pre[len] \*= pre[len - 1];

}

}

int getProduct(int k) {

if (len < k) return 0;

return pre[len] / pre[len - k];

}

private:

int len, pre[40010];

};

/\*\*

\* Your ProductOfNumbers object will be instantiated and called as such:

\* ProductOfNumbers\* obj = new ProductOfNumbers();

\* obj->add(num);

\* int param\_2 = obj->getProduct(k);

\*/

复杂度分析：

时间复杂度：add和getProduct复杂度均为O(1)。

空间复杂度：O(n)，需要额外提供一个辅助数组。