**难度: 简单**

你现在是一场采特殊赛制棒球比赛的记录员。这场比赛由若干回合组成，过去几回合的得分可能会影响以后几回合的得分。

比赛开始时，记录是空白的。你会得到一个记录操作的字符串列表 ops，其中 ops[i] 是你需要记录的第 i 项操作，ops 遵循下述规则：

整数 x - 表示本回合新获得分数 x

"+" - 表示本回合新获得的得分是前两次得分的总和。题目数据保证记录此操作时前面总是存在两个有效的分数。

"D" - 表示本回合新获得的得分是前一次得分的两倍。题目数据保证记录此操作时前面总是存在一个有效的分数。

"C" - 表示前一次得分无效，将其从记录中移除。题目数据保证记录此操作时前面总是存在一个有效的分数。

请你返回记录中所有得分的总和。

示例 1：

输入：ops = ["5","2","C","D","+"]

输出：30

解释：

"5" - 记录加 5 ，记录现在是 [5]

"2" - 记录加 2 ，记录现在是 [5, 2]

"C" - 使前一次得分的记录无效并将其移除，记录现在是 [5].

"D" - 记录加 2 \* 5 = 10 ，记录现在是 [5, 10].

"+" - 记录加 5 + 10 = 15 ，记录现在是 [5, 10, 15].

所有得分的总和 5 + 10 + 15 = 30

示例 2：

输入：ops = ["5","-2","4","C","D","9","+","+"]

输出：27

解释：

"5" - 记录加 5 ，记录现在是 [5]

"-2" - 记录加 -2 ，记录现在是 [5, -2]

"4" - 记录加 4 ，记录现在是 [5, -2, 4]

"C" - 使前一次得分的记录无效并将其移除，记录现在是 [5, -2]

"D" - 记录加 2 \* -2 = -4 ，记录现在是 [5, -2, -4]

"9" - 记录加 9 ，记录现在是 [5, -2, -4, 9]

"+" - 记录加 -4 + 9 = 5 ，记录现在是 [5, -2, -4, 9, 5]

"+" - 记录加 9 + 5 = 14 ，记录现在是 [5, -2, -4, 9, 5, 14]

所有得分的总和 5 + -2 + -4 + 9 + 5 + 14 = 27

示例 3：

输入：ops = ["1"]

输出：1

提示：

1 <= ops.length <= 1000

ops[i] 为 "C"、"D"、"+"，或者一个表示整数的字符串。整数范围是 [-3 \* 104, 3 \* 104]

对于 "+" 操作，题目数据保证记录此操作时前面总是存在两个有效的分数

对于 "C" 和 "D" 操作，题目数据保证记录此操作时前面总是存在一个有效的分数

来源：力扣（LeetCode）

链接：https://leetcode-cn.com/problems/baseball-game

著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。

(这道题我是点栈进来做的,没用使用到栈)

**思路1(使用的for+switch):**

1). 使用for循环来遍历字符串数组,然后用过switch来选择对应的,数字,C,D,+,

2). 然后按要求进行操作即可(代码有注释)

|  |
| --- |
| public int calPoints(String[] ops) {  //暂时存放的结果集  int[] tempRes = new int[ops.length];  int index = 0; //暂时存放结果集的下标  int res = 0; // 存放真正结果  for (int i = 0; i < ops.length; i++) {  switch(ops[i]) {  case "C" :  //移除上一位  index--;  res -= tempRes[index];  break;  case "D":  //表示上一次的两倍  tempRes[index] = tempRes[index -1] \* 2;  res += tempRes[index];  index++;  break;  case "+" :  tempRes[index] = tempRes[index-2] + tempRes[index-1];  res += tempRes[index];  index++;  break;  //默认是数字  default:  tempRes[index] = Integer.parseInt(ops[i]);  res += tempRes[index];  index++;  }  }  return res;  } |

**思路2(使用栈):**

**使用栈效率比使用方法1低,代码也挺好理解的**

|  |
| --- |
| public int calPoints2(String[] ops) {  Deque<Integer> stack = new ArrayDeque<>();  int res = 0;  for (int i = 0; i < ops.length; i++) {  switch(ops[i]) {  case "C" :  //移除上一位  res -= stack.pop();  break;  case "D":  //表示上一次的两倍  res += stack.peek()\*2;  stack.push(stack.peek()\*2);  break;  case "+" :  int beforeValue = stack.pop();  int newValue = beforeValue + stack.peek();  res += newValue;  stack.push(beforeValue);  stack.push(newValue);  break;  //默认是数字  default:  res += Integer.parseInt(ops[i]);  stack.push(Integer.parseInt(ops[i]));  }  }  return res;  } |