E1 - Solution

`A` 摩卡当梦拓

难度	考点
1	循环结构 输入输出

题目分析

这道题和 `C1-C-拼接URL` 是类似的，只不过多了一个要处理 n 名同学的循环过程。

分析题目，一开始时先输入一个整数 n，代表学生的数量，也就是说 Moca 有 n 个学生要发短信。然后，会输入 n 个学生的学号和班级号，为这 n 名同学发出 n 条短信，实际上 n 名同学与 n 条短信之间存在一个一一对应的关系，所以我们可以将题目理解为进行 n 次输入一个学生的学号和班级号，再输出一条短信的过程。也就是说，我们有一个执行 n 次的循环，每次循环进行一个输入和输出操作。实际上，这就是这道题的思路。

在比赛的过程中，发现有不少同学纠结于多组输出一起出现，实际上，对于 OJ 上的多组数据评测：OJ 中题目经常需要处理多组数据，每组数据给出对应的输出数据，如第一次作业赛的 A，这些题目中每组数据完全独立，直接根据每组输入计算对应的输出。 对于此类题目，代码处理时并不需要一次性读入全部的输出，再逐个计算后输出，只需要一个个计算和输出。此时，本地环境下，输入一组数据后，会直接输出结果，再进行下一次输入，即输入和输出会交错显示在一起，但这不影响 OJ 的评测，在 OJ 中，输入和输出是存放在不同的文件中，评测机只检查输出结果，中间的输入不会影响评测。

其次就是不要自己手敲让输出的格式串，直接复制就好了。

示例代码

```c
#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n);

    for(int i = 1; i <= n ;i++) {
        int id, c;
        scanf("%d%d", &id, &c);
        printf("Greetings from BUAA, your Student ID is %d, Class Number is %d.\n", id, c);
    }
    return 0;
}
```

`B` cyz 买文具

难度	考点
1	基本输入输出和简单计算

题目分析

已知钱的总数是 x 元 y 角 z 分，那么以分为单位，可以求得总钱数等价于 $x*100+y*10+z$

接下来用总数除以259便可以得到最多买笔的个数

示例代码

```
#include<stdio.h>
int x,y,z;
int main()
{
    scanf("%d%d%d",&x,&y,&z);
    printf("%d",(x*100+y*10+z)/259);    //向下取整就是正常的"/"

    return 0;
}
```

`C` 摩卡与小学期

难度	考点
2	分支结构

题目分析

根据题目要求，我们先要对预先给定个数的 `贡献值` 求和，然后根据题目描述和样例，进行一个判断：如果 `贡献值` 之和小于等于 0， 则 Moca 团队一个需求也没有完成；如果 `贡献值` 之和大于等于 `老师发布的需求数`，那么 Moca 团队完成了老师发布的全部需求；否则 Moca 团队完成了与贡献值之和相等的需求。

此题需要注意的就是一些特殊的数据点，这里故意设了几个数据点，包括 Moca 团队的贡献值之和**恰好等于** 0，Moca 团队的贡献值之和**恰好等于**老师发布的需求数，这两种情况下按照题意是要输出特殊语句的。有的同学忽略了临界情况，即在分支结构的条件判断中没有写等号，导致没能通过特殊数据点。

其次还有一点需要额外说明的是，一些同学用变量 `sum` 保存贡献值之和，在 `main` 里定义了 `sum` 但是没有初始化，这种情况下会导致 `sum` 的初值不一定是 0，原理在此无法过多赘述。总之，大家一定要注意类似于这样的问题，即应该写 `int sum = 0;`

示例代码

```
#include <stdio.h>

int main()
{
    int sum = 0;
    int n, a, need;
    scanf("%d",&n);
    for(int i = 1;i <= n;i++) {
        scanf("%d", &a);
        sum = sum + a;
    }
    scanf("%d", &need);

    if( sum <= 0 ) {
        printf("%d\n",0);       // 根据题意，贡献值为负或者为 0 时是一个需求也没完成的情况
        printf("Moca finish 0 requirement!");
    }
    else if( sum >= need ) {
        printf("%d\n",need);    // 根据题意，贡献值大于等于老师发布的需求则完成了所有需求
        printf("Moca finish all requirements!");
    }
    else {
        printf("%d\n",sum);
    }

    return 0;
}
```

`D` 鸡兔同笼2025

难度	考点
2	解方程，if判断

题目分析

本题只需要解二元一次方程 $\begin{cases} x+y=n \\ 2x+4y=m \end{cases}$ ，并判断解得的 x, y 是否满足以下两个要求：

- x, y 都是非负整数。
- 由于求解过程使用整除，需判断 x, y 带入原方程时是否能使等式成立。

只有满足以上两个条件，才算原方程有非负整数解。

示例代码

```
#include <stdio.h>

int main()
{
    int x, y, n, m;
    while (scanf("%d%d", &n, &m) != EOF)
    {
        x = (4 * n - m) / 2;
        y = n - x;
        // 判断解是否满足要求
        if (m % 2 != 0)
            printf("No answer\n");
        else if(x < 0)
            printf("No answer\n");
        else if(y < 0)
            printf("No answer\n");
        else
            printf("%d %d\n", x, y);
    }

    return 0;
}
```

扩展延伸

当我们学到函数时，我们就会理解 `scanf("%d%d", &n, &m) != EOF` 代表什么含义，但是不定组输入大家一开始就应该掌握。现在我们只需要知道利用这个代码，就可以处理不定组输入，`scanf` 会一直读下去，直到输入结束或主动跳出循环。

E 分类三角形

难度	考点
2	if条件判断、简单的三角形划分

题目分析

为了在进行三角形判断时方便，可以采取按边的大小排序

- 三角形任意两边之和大于第三边，因此只要两个较短边之和不大于最大边便无法组成三角形
- 根据余弦定理判定，当两条较短边平方和等于第三边为直角三角形，小于第三边为钝角三角形，大于第三边为锐角三角形
- 只要存在两条边长度相等即为等腰三角形，可以用一个整数来标记是不是有两条边相等（其实只要比较中间的边和最大最小边是否相等就可以）
- 满足三边同时相等即为等边三角形（只要比较最长边和最短边是否相等就可以）

示例代码

```
#include<stdio.h>
int a,b,c;
int main()
{
    scanf("%d%d%d",&a,&b,&c);
    int t;

    // 将边按长度排序，c 是最大的，b 次之，a 是最小的
    if(a>b) t=a,a=b,b=t;     // 这段代码可以实现交换 a 和 b 的值，可以手动模拟一下
    if(a>c) t=a,a=c,c=t;
    if(b>c) t=b,b=c,c=t;

    if(a+b<=c) printf("Not triangle\n");
    else{
        if(a*a+b*b==c*c) printf("Right triangle\n");
        else if(a*a+b*b<c*c) printf("Obtuse triangle\n");
        else printf("Acute triangle\n");

        // 判断等腰三角形
        // 或者用 if - else if
        int tem = 0;
        if(a==b) tem = 1;
        if(b==c) tem = 1;
        if(tem == 1) printf("Isosceles triangle\n");
```

```
        // 判断等边三角形
        if(a==c) printf("Equilateral triangle\n");
    }

    return 0;
}
```

F 检查 ID

难度	考点
3	循环，判断。

题目分析

题目要求我们检查输入的 id 的每一位，首先注意到，对于一个数字 x，$x\%10$ 就是 x 的最后一位的值。我们又注意到 $\frac{x}{10}$ 就是 x 去掉最后一位的值，所以结合使用这两种运算直到检查完 x 的每一位，此时 x 会变成 0 ,此时循环结束。

我们可以在检查到 6 之后输出 `Ban lP` ，然后直接结束程序，避免接下来已经没有意义的继续判断；如果我们成功离开了循环，证明输入的数字中没有 `6`，此时输出 `Xhesica win` 。通过 `return 0;` ，可以实现直接结束程序的目的。

示例代码

```
#include<stdio.h>
int x;
int main(){
    scanf("%d",&x);
    while(x != 0){
        if(x%10 == 6) {
            printf("Ban lP");
            return 0;
        }
        x = x / 10;
    }
    printf("Xhesica win");
    return 0;
}
```

G yet another a+b problem

难度	考点
4	取模运算

题目分析

令 $c = a + b$。

考虑 $c \geq 0$ 的时候直接取模即可。

$c < 0$ 时仍然对 p 取模，求得负数意义下的模数，此时 $-p < c \, mod \, p \leq 0$，再加一个 p 即可，把两种情况综合一下就是 $((c\%p)+p)\%p$ 了。

建议大家平时写代码时 if 语句能少就少，能综合考虑的尽量综合考虑。

注意在运算过程中不要超过 int 上限。

示例代码 1

```
#include<stdio.h>
int main(){
    int a,b,p;
    scanf("%d%d%d",&a,&b,&p);
    if(a + b >= 0) {
        printf("%d", (a + b) % p);
    } else {
        printf("%d", (a + b) % p + p);
    }

    return 0;
}
```

示例代码 2

```
#include<stdio.h>
// 想想看，为什么这个代码与上面的代码一样
int main(){
    int a,b,p;
    scanf("%d%d%d",&a,&b,&p);
    printf("%d",((a+b)%p+p)%p);

    return 0;
}
```

扩展延伸

这里补充一下 C 语言的整数除法和取余运算，我们知道比如在同余的角度下，$4 \bmod 3 = 1$ 或者 $4 \bmod 3 = -2$ 都可以说是对的，那么 C 语言是如何规定取余操作的呢？可以尝试以下的代码片段：

```c
#include <stdio.h>

int main()
{
    printf("4 %% 3 = %d\n",4 % 3);
    printf("-2 %% 3 = %d\n",(-2) % 3);
    printf("-5 %% 3 = %d\n",(-5) % 3);
    printf("(-2 + 3) %% 3 = %d\n",(-2+3) % 3);
    printf("(-5 + 3) %% 3 = %d\n",(-5+3) % 3);
    printf("(-2 %% 3 + 3) %% 3 = %d\n",(-2%3 + 3) % 3);
    printf("(-5 %% 3 + 3) %% 3 = %d\n",(-5%3 + 3) % 3);

    printf("\n");

    printf("4 %% -3 = %d\n",4 % -3);
    printf("-2 %% -3 = %d\n",(-2) % -3);
    printf("-5 %% -3 = %d\n",(-5) % -3);
    printf("(-2 + 3) %% -3 = %d\n",(-2+3) % -3);
    printf("(-5 + 3) %% -3 = %d\n",(-5+3) % -3);
    printf("(-2 %% -3 + 3) %% -3 = %d\n",(-2%3 + 3) % -3);
    printf("(-5 %% -3 + 3) %% -3 = %d\n",(-5%3 + 3) % -3);
    return 0;
}
```

在 C 语言中，余数的计算方法可以表示为 $a \bmod b = a - a/b * b$ 。而 C 语言中除法是趋零截尾的，也就是整数除法的结果相当于除法的结果向 0 的方向，将小数部分阶段，比如计算整数除法 $7/(-3)$ ，$\frac{7}{-3} = -2.33$ ，所以整数除法结果就是 -2 ，即直接截断了小数点后的部分。

所以余数和被除数的符号一致，也就是说余数并不总是和通常的取余运算结果相一致（通常数学上的取余操作 $a \bmod b$ 结果在 0 到 $b-1$），所以在执行完 `ans = a % b` 后，为了使结果 ans 一定在 0 到 $b-1$ 之间，我们用 `ans = (ans + b) % b`。这样，如果 ans 以前的值就是一个正数，那么 ans 的值不会受到影响；如果 ans 的值之前是一个负数，这么做就相当于把它转化成了同余意义下等价的正数，符合我们通常数学中的取余操作。

H shtog 挑武器

难度	考点
4	循环结构，分支结构，求最值

题目分析

这道题就是输入若干个数去求他们的最大值，那么我们很容易可以想到从头到尾遍历每个数，同时记录一下已经遍历过的数中最大的数，当遍历到新的数的时候，就和这个已经遍历过的最大值比较一下，如果比已有的最大值更大，那么就将最大值更新为新的值，同时用一个变量记录下来这个新的最大值对应的编号，否则继续遍历。

同时要注意，题目要求有多个最大值时要输出编号最大的，也就是靠后的数。因此如果遍历到了一个值和已有的最大值一样大，同样要将最大值的编号更新为这个靠后的数的编号。

示例代码

```c
#include <stdio.h>
int main() {
    int n, t, maxV = -1, index; // maxV记录遍历过的数的最大值，用于比较. index 记录最大值的编号
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%d", &t);
        if (t >= maxV) maxV = t, index = i + 1; // 这里是 >= 符号，因为等于的时候也应该更新index
    }
    printf("%d", index);

    return 0;
}
```

`I` 小P的乌龟爬水井 2

难度	考点
5	贪心、分类

题目分析

下面是 `C1 - H - 小P的乌龟爬水井` 的题解

由题，我们可以发现，当 $a \geq h$ 时，乌龟总能在第 1 天到井口，故可将此类情况单独分出。

之后，当 $a < h$ 时，可再分为两种情况。

1. 当 $a \leq b$ 时，由于乌龟上升距离不大于下降距离，故其每天开始时总位于井底。又由于其不能一次爬到井口，即 $a < h$ ，故其始终不能到达井口。

2. 当 $a > b$ 时，由于乌龟上升距离大于下降距离，故其每天的开始高度总比上一天高。故而，其总能到达井口。并且，由于夜晚的高度会下降，故乌龟一定在最后一天的白天就到达了终点。设第 ans 到达了井口，则一定有 $(ans-1)*(a-b)+a \geq h$ 其中 $(ans-1)*(a-b)$ 代表前 $ans-1$ 天行进的路程，而 a 代表最后一天行进的路程，二者相加应大于 h。变形可得 $ans \geq (h-a)/(a-b)+1$，即 $ans = \lceil (h-a)/(a-b) \rceil + 1$ ，直接计算即可。

以上为 `C1 - H` 的题解，下面将讲述如何将其他范围的 a 和 b 转换为只有自然数范围的 a 和 b。

1. $a < 0, b \geq 0$ 时：由于乌龟两次回落均在同一天，故若将其视为一次回落并不会影响最后结果。因而可将两次回落进行合并，并将向上爬距离视为 $0cm$。即令 $a_1 = 0, b_1 = b - a$，此时等价于乌龟白天向上爬 a_1cm，晚上向下爬 b_1cm，之后便可转换为 $C1H$ 的程序进行计算。
2. $a \geq 0, b < 0$ 时：同理，由于乌龟两次向上爬均在同一天，故若将其视为一次向上爬并不会影响最后结果。因而可将两次向上爬进行合并，并将回落距离视为 $0cm$。即令 $a_1 = a - b, b_1 = 0$，此时等价于乌龟白天向上爬 a_1cm，晚上向下爬 b_1cm，之后便可转换为 $C1H$ 的程序进行计算。
3. $a < 0, b < 0$ 时：有两点要提前说明：由于乌龟最低位于井底，不会位于更低的地方，故而第一天白天的回落没有意义，可视为从第一天晚上的上升开始计算；因为此时为晚上上升，白天下降，故而乌龟总会于夜晚到达井口。在当前状况下的一天中，回落总是先于上爬出现的。但是因为第一天白天没有意义，若将每一天的夜晚与后一天的白天当作一天看待，即总体向前平移半天，便会发现其又变成 $C1H$ 的先上升后下降了。并且，因为总是在夜晚到达，在时间平移后仍在同一天到达井口，只有白天与夜晚的差别，故而二者答案没有任何差别。因此，可令 $a_1 = -b, b_1 = -a$，此时等价于乌龟白天向上爬 a_1cm，晚上向下爬 b_1cm，之后便可转换为 $C1H$ 的程序进行计算。

所以，不论是哪一种情况，最后都能简单地化为都是正数的情况去解决，当全是负数的时候，相当于 $a_1 = -b$, $b_1 = -a$ 的情况；当 a 正 b 负时，相当于 $a_1 = a - b$, $b_1 = 0$ 的情况；当 a 负 b 正时，相当于 $a_1 = 0$, $b_1 = b - a$ 的情况。希望同学们能学会转化问题的能力，把遇到的新问题转化为自己已经解决的问题。

第二点，发现有不少同学写的时候有诸如这样的代码：`h + a - b` 或者 `(h + a + b) - a` ，实际上，参考 `C1 - I - 军乐团破冰` 的提示，以及 P1 上的例子（十亿加二十亿不等于三十亿，而是一个负数）；即 `int` 的表示范围是有上限的，大概是 21 亿（准确的值是 `2147483647`），所以前面两个计算会超过 `int` 的表示范围，正确的做法是用诸如 `(a - b) / (a - b) = 1` 和 `(a - a) = 0` 这样的方法合并，避免出现 `h + a - b` 或者 `(h + a + b) - a` 。

PS：出题人将乌龟和蜗牛记混了……

示例代码

```
#include<stdio.h>
signed main()
{
    int t,a,b,h,ans,temp;//此题中int就足够了
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d%d%d",&h,&a,&b);
        //以下为新增代码
        if(a<0)
        {
            if(b<0)//a1=-b b1=-a
            {
                temp=a;
                a=-b;
                b=-temp;
            }
            else//a1=0 b1=b-a
            {
                b=b-a;
                a=0;
            }
        }
        else
        {
            if(b<0)//a1=a-b b1=0
            {
                a=a-b;
                b=0;
            }
        }
        //以上为新增代码
        if(a>=h)//a>=h时，第一天白天就可到顶，与b无关
        {
            printf("1\n");
        }
        else if(a<=b)//a<h且a<=b时，总无法到顶
        {
            printf("Impossible\n");
        }
        else//b<a<h时，可通过表达式计算出天数
        {
            ans=(h-b-1)/(a-b)+1;//a/b向上取整等价于(a+b-1)/b的向下取整
            printf("%d\n",ans);
        }
    }
```

```
    return 0;
}
```

J 摩卡与探险家水獭

难度	考点
4	数学

题目分析

假设水獭从 $(0,0)$ 去往 (m,n) 点：

- **如果 m 是** 0： 很显然经过的整数点数是 $n-1$ （$n=0$ 时除外）
- **如果 n 是** 0： 很显然经过的整数点数是 $m-1$ （$m=0$ 时除外）
- 否则，那么水獭所走的直线的方程为 $y=\frac{n}{m}x$，如果 $gcd(n,m)\neq 1$，那么显然该直线方程不是最简形式，可将直线方程上下同除 $gcd(n,m)$，将直线方程化为最简形式 $y=\frac{n'}{m'}x$ ，那么问题就变成了有多少 i，满足 $i<m$，且 i 是 m' 的倍数；其答案的个数为 $\frac{m}{m'}-1$，而 $m'=\frac{m}{gcd(n,m)}$，所以答案的个数即为 $gcd(n,m)-1$ 。

首先我们需要暴力地去枚举每一个位置，规模大概是 $O(MN)$，由于 M 和 N 的数据范围相当，所以时间复杂度为 $O(N^2)$。求 gcd 时，如果我们用暴力枚举判断模数是否等于 0 的方法，总共大约，总的时间复杂度大概为 $O(N^3)$，而 N 最大为 1000，这样的方法无疑效率太低了。于是，我们可以利用高中学过的辗转相除法（欧几里得算法）来求 gcd，求 gcd 的效率将会大大提高，总的时间复杂度大概为 $O(N^2logN)$ 。

示例代码

```
#include <stdio.h>

int main() {
    int m, n;
    scanf("%d%d", &m, &n);
    int ans = 0;

    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            // 扣去 (0, 0) 点
            if (i + j != 0) {
                int a = i;
                int b = j;
                // 辗转相除法计算 gcd
                while (b != 0) {
                    int temp = b;
```

```
                    b = a % b;
                    a = temp;
                }
                // 现在 a 是 gcd(i, j)
                ans = ans + a - 1;
            }
        }
    }

    printf("%d", ans);

    return 0;
}
```

扩展延伸

时间复杂度：[复杂度 - OI Wiki]

辗转相除法时间复杂度：[辗转相除法的时间复杂度log(N)的简洁证明]

下面再提供两种时间复杂度更低的方法供同学们参考，此部分内容**完全不需要同学们掌握**，同学们了解。

以下内容均由 **Saisyc** 提供：

扩展方法一

欧拉函数（Euler function）：

$$\varphi(n) = n \prod_{\text{素数}p|n} \frac{p-1}{p}$$

有结论：

$$\sum_{\substack{1\leqslant i\leqslant m \\ 1\leqslant j\leqslant n}} \gcd(i,j) = \sum_{k\in\mathbf{N}_+} \left\lfloor \frac{m}{k} \right\rfloor \left\lfloor \frac{n}{k} \right\rfloor \varphi(k)$$

[结论的证明]

令 $x = \max\{m,n\}$ 可以证明如下算法的时间复杂度是 $\mathcal{O}\left(\frac{x\sqrt{x}}{\ln x}\right)$：

```
#include <stdio.h>
int main() {
    int m, n, result = 0;
    scanf("%d%d", &m, &n);
```

```
    for (int i = 2; i <= m && i <= n; i++) {
        int u = i, v = i;
        for (int j = 2; j * j <= u; j++) {
            if (u % j == 0) {
                v = v / j * (j - 1);
                while (u % j == 0) {
                    u = u / j;
                }
            }
        }
        if (u > 1) {
            v = v / u * (u - 1);
        }
        result = result + v * (m / i) * (n / i);
    }
    result = result + m * (m - 1) / 2;
    result = result + n * (n - 1) / 2;
    printf("%d", result);
    return 0;
}
```

扩展方法二

扩展方法二用到了后面大家要学习的数组，**仅供参考，不需要掌握**：

我们可以计算每条直线经过的整点数量，然后把不同的直线经过的整点数量加在一起就可以了；当我们统计 (i, j) 所在的直线的整点数量时，我们实际上已经把 $(2i, 2j)$ ，$(3i, 3j)$ ... 这些直线上的整点数量也给统计完了，可以用一个二维数组来标记每个点所在的直线是否被统计过：

```
# include <stdio.h>

int a[1001][1001];
int m, n;

int min(int x, int y) {
    return x < y ? x : y;
}

int main() {
    scanf("%d%d", &m, &n);
    int result = 0;
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (a[i][j] == 0) { // 如果 (0,0) 到 (i,j) 这条直线上的整点没有被计算过
                int t = min(m / i, n / j); // 这条直线上在题目范围内的整点的数量
```

```
                result = result + t * (t - 1) / 2; // 对于这条直线上的整点进行计算
                for (int k = 2; k * i <= m && k * j <= n; k++) {
                    a[k * i][k * j] = 1; // 将这条直线上的整点标记为已计算
                }
            }
        }
    }
    result = result + m * (m - 1) / 2;
    result = result + n * (n - 1) / 2;
    printf("%d", result);
    return 0;
}
```