A 略有出入

题目分析

输入，相减，输出。

```
#include <math.h>
#include <stdio.h>

int main() {
    double a, b;
    scanf("%lf%lf", &a, &b);
    double delta = fabs(a - b);
    double eps = 1e-8;
    if (delta < eps) {
        printf("0");
    } else if (a > b) {
        printf("1");
    } else {
        printf("-1");
    }
}
```

B “我能单刷他们”

题目分析

基本的循环+条件判断。没有分析，直接上代码

示例代码

```
#include <stdio.h>

int main()
{
    int x = 0;
    scanf("%d", &x);

    for (int i = 0; i < 7; i++) {
        int life;
        scanf("%d", &life);
        if (x == life) {
            x += 2;
        }
    }
```

```
    printf("%d", x);
    return 0;
}
```

C 去买花生

题目分析

第一行的输入 x 没有实际作用，并且可以取负值。忽略即可。 以字符串或多个字符识别均可。

希望你没有被这题卡住。

示例代码

```
#include <stdio.h>

int main() {
    int x, ans = 0;
    char s[2000] = {};
    scanf("%d%s", &x, s);
    for (int i = 0; i <= 1005; i++) {
        ans = ans + (s[i] == 'n') + 2 * (s[i] == 'd');
        if (s[i] == 'c')
            break;
    }
    printf("%d", ans);
}
```

或者

```
#include <stdio.h>

int main() {
    int ans = 0;
    char in;
    while (scanf("%c", &in) > 0) {
        ans += (in == 'n') + 2 * (in == 'd');
        if (in == 'c')
            break;
    }
    printf("%d", ans);
}
```

D 重炮的进制转换

题目分析

因为每一位十六进制数字都可以用四位二进制数字表达出来，因此本题只需要读入一位十六进制数字并输出其对应的二进制数字即可。

给个对照表，便于大家理解：

十六进制	二进制	十六进制	二进制	十六进制	二进制	十六进制	二进制
0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

本题有多种做法，这里给出使用位运算的方法。

思路分析：

对每一位十六进制数字，首先将读入的字符型变量转换成对应的整型变量，然后依次取出该整型变量的二进制表示的后四位并输出即可。

标准程序：

```
#include <stdio.h>

int main(){
    char c;
    int num;
    while(scanf("%c", &c) != EOF){
        if(c >= '0' && c <= '9'){
            num = c - '0';       //将字符型变量转换成对应的整型变量
            for(int i = 3; i >= 0; i--)
                printf("%d", (num >> i) & 1);    //从高到低依次取出整型变量的后四位即可。
        }
        if(c >= 'a' && c <= 'f'){//过程同上
            num = c - 'a' + 10;
            for(int i = 3; i >= 0; i--)
                printf("%d", (num >> i) & 1);
        }
    }
    return 0;
}
```

E 十进制数按位与

题目分析

把每个十进制位分开，按位求最小值，输出。

你也可以选择使用字符串的做法，假装是字符串题，从而回避int的操作。

不要忘记妥善处理各种0的情况。

示例代码

```
#include <stdio.h>

#define max(a, b) (a > b) ? a : b
#define min(a, b) (a < b) ? a : b

int main() {
    int n, m, ans[20]={};
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= 19; i++) {
        ans[i] = min((n % 10), (m % 10));
        n /= 10, m /= 10;
    }
    int j = 15;
    while (ans[j] == 0 && j != 1) j--;
    for (; j >= 1; j--) {
        printf("%d", ans[j]);
    }
}
```

F 威廉的硬币翻转

题目分析

让我们回顾一下题目中“一次操作”的过程：

- 每次都翻动最右边的硬币
- 如果一个硬币从正面翻回反面，那么将其左侧紧邻的硬币翻面。

如果把一串硬币用 0 和 1 组成的数列表示， 0 表示反面， 1 表示正面，显然进行一次操作就是把这样表示出来的一个二进制数加一。因此题目只需要求得 N 的二进制表达中 1 的个数即可。有很多种求的方式，这里给出一种基于位运算的计算方法。

思路分析：

每次取末位后将数右移一位即可。如果不理解可以自己在草稿纸上推演一下。

标准程序：

```c
#include <stdio.h>

int main(){
    long long N;
    scanf("%lld", &N);
    int cnt = 0;
    while (N){
        if (N & 1 == 1)cnt++;
        N >>= 1;
    }
    printf("%d", cnt);
    return 0;
}
```

G 二舅也许能治好我的精神内耗

题目分析

本题考察位运算的性质和构造思维，所以遍历是一定不行的。我们先考虑最后一位，发现三个数的末尾不管是3个都是1，或者两个1一个0，一个1两个0，三个数相加必为偶数，所以奇数无解。

那么我们从偶数的角度考虑，首先明确$0 \oplus x = x$,所以我们只要两个0，一个数是$n/2$即可满足条件。

参考代码

```c
include<stdio.h>
int main()
{
    int t,n;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        if(n%2==1){
            printf("-1\n");
        }else{
            printf("%d 0 0\n",n/2);
        }
    }
    return 0;
}
```

H 朝田诗乃的分组加密

其实本题还是比较简单的，只要把握好我们每一步需要用的量的就可以。

读入密钥和明文之后，将明文按照 8 个数字一组分类，然后依次处理即可。

```c
#include<stdio.h>
```

```
int n;
int IV[10],Plaintext[1000020],Ciphertext[10];
int main(){
    scanf("%d",&n);
    for(int i=1; i<=8; i++){
        scanf("%d",&IV[i]);
    }
    for(int i=1; i<=n*8; i++){
        scanf("%d",&Plaintext[i]);
    }
    for(int i=0; i<n; i++){
        int One_test = 0, X_test = 0;
        for(int j=1; j<=8; j++){
            Ciphertext[j] = IV[j] ^ Plaintext[i*8+j];
            int a = Plaintext[i*8+j];
            while(a){
                if(a%2==1) One_test++;
                a >>= 1;
            }a = Plaintext[i*8+j];
            while(a){
                if((a%16)>9) X_test++;
                a = a/16;
            }
        }
        for(int j=1; j<=8; j++) printf("%04X",Ciphertext[j]);
        printf(" %d %d\n",One_test,X_test);
        for(int j=1; j<=8; j++) IV[j] = Ciphertext[j];
    }
    return 0;
}
```

I 狼了个狼

题目分析

首先先介绍下[惟一分解定理]：任何一个大于1的整数n都可以唯一地分解成若干个质因数的连乘积。即$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$,其中$p_1, p_2, \cdots, p_k$均为$n$的质因数。此处不予证明，我们直接在此基础上进行分析。

先分析两个操作：

- 增加整数$k-1$倍：即n变为$kn = kp_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$.
- 数量开根：即n变为：$\sqrt{n} = p_1^{\alpha_1/2} p_2^{\alpha_2/2} \cdots p_k^{\alpha_k/2}$,因为此时$n$是完全平方数，所以开根后仍为整数。

此时我们可以发现最关键的突破口：**操作前后相乘的质因子的种类不变！**

因此我们得到结论：$n_{min} \geq p_1 p_2 \cdots p_k$，但我们是否能取到$p_1 p_2 \cdots p_k$呢？答案是肯定的，下面给出证明：

> $\because$ 对于$\forall n$,总$\exists m$,满足$2^m \geq max\{\alpha_1, \alpha_2, \cdots, \alpha_k\}$
> $\therefore$ 可以通过有限次的乘法$q_i n$,使$\alpha_i = 2^m, \forall 1 \leq i \leq k$
> 然后再通过m次开根，使α_i变为1，即$n = q_1 q_2 \cdots q_k$

下面来解决第二个问题：最少需要多少次操作?

要解决这个问题首先要发现操作是满足交换律的（如果交换后可以操作的话），过程如下：

$$n \to \sqrt{n} \to x\sqrt{n} \text{等效于} n \to x^2 n \to x\sqrt{n}$$

因此我们始终可以先进行操作一再进行操作二，所以可以将所有的操作一合并在一次完成，然后再进行m次操作二即可。

注：存在不需要操作一的情况，即所有α_i均等于2^m。

代码思路

- 质因数分解得到最小数量。
- 找到满足$2^m \geq \alpha_i$的最小m。
- 判断是否需要进行操作一。

示例代码

```
#include <stdio.h>

int a[1000], x, n, i;
int main() {
    while (~scanf("%d", &n)) {
        int r = 1, m = 0, cnt = 0;
        for (i = 2; i * i <= n; i ++ )
            if (n % i == 0) { // 质因数分解
                int c = 0;
                while (n % i == 0)
                    n /= i, c ++ ;
                r *= i;
                a[cnt++] = c;
                while (1 << m < c) // 找最小m
                    m ++ ;
            }

        if (n > 1) {
            r *= n;
            a[cnt++] = 1;
            while (1 << m < 1)
                m ++ ;
        }

        for (i = 0, x = a[i]; i < cnt; x = a[++i])
            if (x < 1 << m) { // 判断是否需要操作一
                m ++ ;
                break;
            }
        printf("%d %d\n", r, m);
    }
    return 0;
}
```

J ACM组队

难度	考点
3	位运算

题目分析

本次灵感来自 “寻找单身狗”（21级程序设计基础E8-J）

~~别问出题人为什么把这个题放在了E3，因为出题人也不知道是谁把这个题放出来的，但是这个题确实只要有思路就简单，没有码量~~

“寻找单身狗” 和本题差不多，只不过一个数字要么出现2次，要么出现1次，然后找出唯一只出现了1次的数字（即单身狗）

“寻找单身狗” 一题的思路很有意思，注意到 $\forall x(x \oplus x = 0)$ （其中 $\oplus$ 表示按位异或），另外异或也满足交换律和结合律。 所以把所有的数字都异或起来，就可以找到单身狗。 （当然我记得那个题有个坑点，似乎是可能没有单身狗，但是异或等于0的时候也可能是有一个单身狗是0，所以还需要一个简单的特判）

然而本题实际上思路和那个题是一样的。只需要发明一个所谓*三进制异或*就行了。

什么意思呢？一种做法就是把输入的数字转换为3进制，然后分别统计每一个三进制位的和模3的余数，最后得到的3进制数再转换成10进制即可。 （见标程1）

~~怎么样？是不是全都在E3的范围内呢？~~

当然上面只是一种做法，实际上，你可以发现转换成2进制也可以，只需要统计每一位的和模3的余数，再将原数转换为10进制就行了。 （见标程2）

~~想看出题人的吐槽吗？欢迎阅读示例程序3~~

示例程序1

```
#include <stdio.h>

int k;
int val[100];

int main()
{
    scanf("%d", &k);
    k = 3 * k + 1;
    for(int i = 0; i < k; i++)
    {
        int r; scanf("%d", &r);
        for(int i = 0; r; i++)
        {
            val[i] += (r % 3); // 转换3进制！并把这一位加到第i为中
            r /= 3;
        }
```

```
    }
    int x = 0;
    for(int i = 30 /*19足够了，不过大一点没有什么关系（毕竟难得算）*/; i >= 0; i--)
        x = x * 3 + val[i] % 3; // 把3进制转换回来
    printf("%d\n", x);
    return 0;
}
```

示例程序2

```
#include <stdio.h>

int k;
int val[100];

int main()
{
    scanf("%d", &k);
    k = 3 * k + 1;
    for(int i = 0; i < k; i++)
    {
        int r; scanf("%d", &r);
        for(int i = 0; r /* 和 r != 0 是一个意思 */; i++)
        {
            val[i] += r & 1; // 转换2进制！并把这一位加到第i为中
            r >>= 1; // 转换2进制就可以大量使用位运算
        }
    }
    int x = 0;
    for(int i = 40 /*31足够了，不过大一点没有什么关系（毕竟难得算）*/; i >= 0; i--)
        x = (x << 1) + (val[i] % 3); // 把2进制转换回来
    printf("%d\n", x);
    return 0;
}
```

示例程序3

```
/*
 * Author: Toby
 * Copyright (C) 2022-2022 Toby Shi
 * （请自行忽视上面一句开玩笑的版权声明）
 * 这是出题人引以为傲的位运算版本
 * 但是不知道为什么没有快多少
 * 果然还是scanf太慢了吗？
 * 早知道写个快读
 * 不然就不至于连qsort都卡不掉了
 * 可恶啊，去年那个题我可是写了一个快读+基数排序+循环展开才卡掉的
 * （暴风哭泣）
 * 这个程序就不写注释
 * 有兴趣的同学自行研究吧
 */
```

```c
#include <stdio.h>

int k;
int x, y, xx, yy;

int main()
{
    scanf("%d", &k);
    k = 3 * k + 1;
    for(int i = 0; i < k; i++)
    {
        int r; scanf("%d", &r);
        xx = x; yy = y;
        x = xx ^ r ^ (yy & r);
        y = yy ^ ((xx | yy) & r);
    }
    printf("%d\n", x);
    return 0;
}
```