# C4 - Solution

## A 摩卡与 lcm

| 难度 | 考点 |
| --- | --- |
| 1 | gcd |

## 题目分析

根据公式 $lcm(a,b)=\frac{a\times b}{gcd(a,b)}$ 计算即可，其中 $gcd(a,b)$ 要利用 hint 给出的（课上所讲的）辗转相除法计算，否则会超时。

还需要注意的一点是 $a\times b$ 可能超出 `int`，所以做乘法时要做 `long long` 的乘法（可以直接声明为 `long long`，也可以 `1ll * a * b`），避免溢出。

## 示例代码

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

int main() {
    int t ;
    scanf("%d", &t);
    for(int i = 1; i <= t ;i++) {
        long long a, b;
        scanf("%lld%lld", &a, &b);
        long long mul = a * b;
        while(b != 0) {
            long long tem = b;
            b = a % b;
            a = tem;
        }
        printf("%lld\n", mul / a);
    }
    return 0;
}
```

## B 庄园地图生成

| 难度 | 考点 |
| --- | --- |
| 2 | 位运算 |

## 题目分析

本题只需要循环 $n$ 次，每次将读入的数转换成 $m$ 位二进制输出就可以了！所以其实不需要会位运算也是完全可以做的，只不过位运算可以简化程序。那么根据本题的数据范围，选择 `unsigned long long` 来读入每个数，然后从高位到低位依次输出。

## 示例代码

```
#include <stdio.h>
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    while(n --)
    {
        unsigned long long t;
        scanf("%llu", &t);
        for(int i = m - 1;i >= 0;i --)
        {
            if((t >> i & 1) == 1)
            {
                putchar('#');
            }
            else
            {
                putchar('.');
            }
        }
        putchar('\n');
    }
    return 0;
}
```

# C 摩卡与成绩统计 3

| 难度 | 考点 |
| --- | --- |
| 2 | 模拟、数组 |

# 题目分析

我们可以把一个学生相关的数据分为三部分，入学年份 $year$，`A - J` 每道题是否通过，`K` 是否通过。

可以先求出一个 `A - J` 过题数的总和 $sum$，折算出对应的分数 $score$；再根据 `K` 是否通过，如果通过不做处理，未通过将 $score$ 映射到相应的分数；最后看入学年份，做相应的判断处理。

同学们出现问题较多的地方就是 $sum = 0$ 时可能没有做相应的处理；或者循环之间有些变量没有重置。

# 示例代码

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

int score[15] = {0, 15, 30, 45, 60, 75, 90, 95, 99, 100, 100};

int main() {
    int n ;
    scanf("%d", &n);
    for(int i = 1; i <= n ;i++) {
        int year;
        scanf("%d", &year);
        int bound = year == 2024 ? 90 : 60; // 新学的三目表达式
        int pass = 0;
        for(int i = 1; i <= 10 ;i++) {
            int tem = 0;
            scanf("%d", &tem);
            pass += tem;
        }

        int k;
        scanf("%d", &k);
        if(!k) {
            if(pass >= 4)
                pass = 4;
            else
                pass = 0;
        }
        int s = score[pass];

        if(s >= bound)
            printf("Congratulations, you pass the exam and your score is %d.\n",
s);
        else
            printf("You have to take the course.\n");
    }

    return 0;
}
```

## D 小懒獭与日期

| 难度 | 考点 |
| --- | --- |
| 3 | 模拟、日期 |

## 题目分析

根据题目中给出的 `Zeller` 公式计算即可，也可以参考 PPT 上的代码。

易错点有如下两个，第一是 $W$ 按照公式计算出来可能 $< 0$，我们要找到其 $[0, 6]$ 之间同余等价的数字，很显然 $+7$ 就可以（因为 $W$ 的范围一定在 $[-6, 6]$）；第二是我们将 $1$ 月和 $2$ 月转化为前一年的 $13$ 和 $14$ 月时，年份也要对应的变化，并且用变化后的年份计算世纪数和年份。

## 示例代码

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
int main()
{
    int year, month, day;
    while (scanf("%d %d %d", &year,&month,&day) != EOF)
    {
        if (month == 1 || month == 2)//判断month是否为1或2
            year--, month += 12;
        int c = year / 100;
        int y = year - c * 100;
        int week = y + y / 4 + c / 4 - 2 * c + 26 * (month + 1) / 10 + day - 1;
        while (week < 0)
            week += 7;
        week %= 7;
        switch (week)
        {
        case 1:printf("Monday\n"); break;
        case 2:printf("Tuesday\n"); break;
        case 3:printf("Wednesday\n"); break;
        case 4:printf("Thursday\n"); break;
        case 5:printf("Friday\n"); break;
        case 6:printf("Saturday\n"); break;
        case 0:printf("Sunday\n"); break;
        }
    }
    return 0;
}
```

## 扩展延伸

蔡勒(Zeller)公式及其推导

# E 寻找因数 (easy version)

| 难度 | 考点 |
| --- | --- |
| 3 | 质数，循环 |

## 题目分析

本题的 $n$ 较大，如果从小到大一个一个枚举试除会超时。但是我们可以只枚举 $1 \sim \sqrt{n}$ 的较小因数，而 $n$ 的较大因数可以通过枚举出的较小因数计算得到。

我们可以用数组把 $1 \sim \sqrt{n}$ 的因数存储起来。输出时，首先正向遍历数组，将它们输出，接着再反向遍历数组，输出 $n$ 除以它们得到的商。这样我们就可以从小到大输出 $n$ 的全部因数了。

注意处理类似 $36, 49$ 这样的完全平方数，此时 $\sqrt{n}$ 恰好就是它的因数之一，而 $\frac{n}{\sqrt{n}} = \sqrt{n}$，一不小心就会把 $\sqrt{n}$ 输出两遍或者漏掉这个因数。

然后就是对数字 $1$ 的处理，它有且仅有一个因数：$1$。看看你的写法是否能正确输出这个比较特殊的情况。

注意，用于存储因数的数组不能开的太小。$1 \sim \sqrt{n}$ 的因数肯定是不可能超过 $\sqrt{n}$ 个的，所以如果你不确定这个数组应该开多少，那么开 $10^6$ 肯定是完全足够的。事实上，$10^{12}$ 以内的拥有最多因数的数是 $963761198400$，它有 $6720$ 个因数，所以你的数组至少要开 $3360$ 才能存储它小于 $\sqrt{n}$ 的因数。（至于这个数字是怎么得到的，欢迎学有余力的同学思考）

## 示例代码

```
#include <stdio.h>
int ans[20000]; // 至少开 3360 的大小
int top;
int main(void)
{
    int t;
    long long n;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%lld", &n);
        top = 0;
        for (int i = 1; 1ll * i * i <= n; i++) // 这里相乘的时候记得强制转换成long
long，否则会溢出
        {
            if (n % i != 0)
                continue;
            ans[top++] = i;
        }
        if (1ll * ans[top - 1] * ans[top - 1] == n) // n 是完全平方数的特殊处理
        {
            printf("%d\n", top * 2 - 1);
        }
        else
        {
            printf("%d\n", top * 2);
        }
        for (int i = 0; i < top; i++)
        {
            printf("%d ", ans[i]);
        }
        for (int i = top - 1; i >= 0; i--)
        {
            if (1ll * ans[i] * ans[i] != n)
                printf("%lld ", n / ans[i]);
        }
        printf("\n");
    }
    return 0;
}
```

# F 小水獭与 AC

| 难度 | 考点 |
| --- | --- |
| 4 | 循环、找规律 |

## 题目分析

本题是典型的 "字符画"，要求在屏幕上打印出某个图案，且图案的规模需要根据输入的数据来决定。

可以分为下面五部分：

- 第一行：输出 $2k$ 个空格，$2k-1$ 个 `*`，再输出 $2k+1$ 个空格，$4k-1$ 个 `*`
- 第 $2$ 至 $k$ 行：对于 `A` 中的两个 `*`，之前输出 $2k-i+1$ 个空格，中间用空格隔开，最后再输出 $2k-i+1$ 个空格；再隔一个空格后输出 `C` 的一个 `*`
- 第 $k+1$ 行：输出的空格与前面类似，输出 $4k-1$ 个 `*`；`C` 的情况与前面类似
- 第 $k+2$ 至 $2k$ 行：与第 $2$ 至 $k$ 行类似
- 第 $2k+1$ 行：`A` 的情况与前面类似，`C` 的情况与第一行类似

## 示例代码

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

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

    // 输出第一行
    for(int i = 1; i <= 2 * k ;i++)
        printf(" ");
    for(int i = 1; i <= 2 * k - 1 ;i++)
        printf("*");
    for(int i = 1; i <= 2 * k + 1 ;i++)
        printf(" ");
    for(int i = 1; i <= 4 * k - 1 ;i++)
        printf("*");
    printf("\n");

    // 输出第 2 - k 行
    for(int i = 2; i <= k ;i++) {
        for(int j = 1; j <= 2 * k - i + 1 ;j++)
            printf(" ");
        printf("*");
        for(int j = 1; j <= 2 * i + 2 * k - 5 ;j++)
            printf(" ");
        printf("*");
        for(int j = 1; j <= 2 * k - i + 2 ;j++)
            printf(" ");
        printf("*");
```

```
        printf("\n");
    }

    // 输出第 k + 1 行
    for(int j = 1; j <= k ;j++)
        printf(" ");
    for(int i = 1; i <= 4 * k - 1 ;i++)
        printf("*");
    for(int j = 1; j <= k + 1 ;j++)
        printf(" ");
    printf("*");
    printf("\n");

    // 输出第 k + 2 - 2k 行
    for(int i = k + 2; i <= 2 * k ;i++) {
        for(int j = 1; j <= 2 * k - i + 1 ;j++)
            printf(" ");
        printf("*");
        for(int j = 1; j <= 2 * i + 2 * k - 5 ;j++)
            printf(" ");
        printf("*");
        for(int j = 1; j <= 2 * k - i + 2 ;j++)
            printf(" ");
        printf("*");
        printf("\n");
    }

    printf("*");
    for(int i = 1; i <= 6 * k - 3 ;i++)
        printf(" ");
    printf("*");
    printf(" ");
    for(int i = 1; i <= 4 * k - 1 ;i++)
        printf("*");
    printf("\n");

    return 0;
}
```

## G 大数变小数

| 难度 | 考点 |
| --- | --- |
| 5 | 字符、循环控制 |

### 题目分析

本题要求一个大数的各位数之和若干次，由于输入的大数有至多 $10^6$ 位数，超出了 $int$ 和 $long\ long$ 能表示的范围，所以只能当一串字符处理。首先可以对于每一个字符计算它与 $'0'$ 的差，将他们加起来就是计算一次各位数字之和的结果，这个结果至多 $9\times10^6$ ，是可以用 $int$ 来表示的，之后再计算各位数字之和就能用这个 $int$ 来算了。

需要注意的是当 $k=0$ 时，必须直接输出 $x$ 的值，以及当 $k$ 特别大时，计算各位数之和会只剩一位数，再继续往下算就都是那一位数了，所以当结果只剩一位数的时候要跳出循环。

## 示例代码

```c
#include <stdio.h>

int main() {
    int n, c;
    long long k;
    scanf("%d %lld ", &n, &k);
    if (k == 0) {
        while ((c = getchar()) != EOF) {
            putchar(c);
        }
        return 0;
    }
    int sum = 0;
    while ((c = getchar()) != EOF) {
        sum += c - '0';
    }
    k--;
    while (k--) {
        if (sum < 10) {
            break;
        }
        int temp = sum;
        sum = 0;
        while (temp) {
            sum += temp % 10;
            temp /= 10;
        }
    }
    printf("%d", sum);
    return 0;
}
```

# H ddz 的大名

| 难度 | 考点 |
| --- | --- |
| 6 | 字符、计数 |

## 题目分析

本题要求一个字符串中 $DDZ$ 作为子序列出现的次数，由于 $DDZ$ 中 $Z$ 只有一个，故我们可以对每个 $Z$ 计算它对答案的贡献。

例如对于字符串中的某个 $Z$，它左边有 $x$ 个 $D$

$\underbrace{DZDD\cdots ZD}_{\text{x 个 D}}\mathbf{Z}\cdots$

从这 $x$ 个 $D$ 中任意选出两个就能和这个 $Z$ 组成一个 $DDZ$，也就是这个 $Z$ 一共能组成 $C_x^2=\frac{x(x-1)}{2}$ 个 $DDZ$，所以只需要从左到右扫一遍，遇到 $D$ 就计数加一，遇到 $Z$ 就计算一下 $C_x^2$，将其加入答案即可。

注意答案可能超出 $int$ 范围，要用 $long\ long$ 存储答案

## 示例代码

```
#include <stdio.h>

int main () {
    int t, c;
    scanf("%d ", &t);
    while (t--) {
        long long res = 0, cnt = 0;
        while ((c = getchar()) != '\n') {
            if (c == 'D') {
                cnt++;
            } else if (c == 'Z') {
                res += cnt * (cnt - 1) / 2;
            }
        }
        printf("%lld\n", res);
    }
    return 0;
}
```

# I 进入虚空

| 难度 | 考点 |
| --- | --- |
| 6 | 多重循环 |

## 题目分析

题意很简单，不做过多介绍。我们首先可以尝试使用暴力方法——枚举所有情况尝试解决。下面是主要代码。

注：下列所有代码总假设数组 $A$ 有效部分从下标 $1$ 开始。

```
for ( int i=1 ; i<=n-2 ; i++ )
    for ( int j=i+1 ; j<=n-1 ; j++ )
        for ( int k=j+1 ; k<=n ; k++ )
            if ( A[i]+A[j]+A[k]==m )
                ans++;
```

提交代码，发现得到了 $0.3$ 分，运行超时。那么我们可以尝试优化这种方法。

因为数组是单调递增的，我们可以发现，如果三元组 $(i_1,j_1,k_1),(i_2,j_2,k_2)$ 满足 $i_1=i_2,A_{j_1}\le A_{j_2}$ ，那么一定有 $A_{k_1}\ge A_{k_2}$ ，从而我们可以让 $k$ 从 $n$ 开始枚举，对于同一个 $i$ ，让 $k$ 只从当前值开始向下枚举，从而减少一部分计算量。下面是主要代码。

```
for ( int i=1 ; i<=n-2 ; i++ )
    for ( int j=i+1,k=n ; j<k ; j++ ){
        for ( ; k>j && A[i]+A[j]+A[k]>m ; k-- );//让 k 一直减到小于等于 j 或者三项和小于等于 m 为止
        if ( j<k && A[i]+A[j]+A[k]==m )
            for ( int l=k ; l>j && A[i]+A[j]+A[l]==m ; l--,ans++ );//因为数组中可能有多个值相同，所以要找到所有满足题意的 k
    }
```

提交代码，得到 $0.9$ 分，运行超时。

再次看一遍代码，发现 $ans$ 每次只加 $1$ 。而最大可能的输出是对于任何三元组都成立的时候，可能有 $C_{3000}^3=4495501000$ 种情况，肯定会超时。（而且这个数也超过了 `int` 能表示的最大范围，所以 $ans$ 应该用 `long long` 存储）

那么遇到这种连续一段数的情况能否优化呢？答案是肯定的。如果连续一段数是相同的，我们就可以通过预处理，得到这一段连续相同值的长度，从而加快计算（注：示例代码中预处理的时间复杂度是线性的）。这样我们就得到了一个比较快的程序，对于每个 $i$ ，数组中每个位置最多只被访问 $1$ 次，时间复杂度是 $O(n^2)$ ，可以通过此题。

需要注意的是，示例代码中如果数组 $A$ 是 `int` 类型，在做加法时有可能溢出。不过因为 $A_i\le 10^9$ ，所以如果溢出了，那么和一定是个负数，不可能与 $m$ 相等，从而对答案没有影响。但这并不意味着我们可以不重视这个问题。我们在编写程序的时候仍然需要注意数据范围，避免溢出。

## 示例代码

```
#include<stdio.h>
int n,m;
long long A[10086],B[10086],ans;
int main(){
    scanf("%d%d",&n,&m);
    for ( int i=1 ; i<=n ; i++ )
        scanf("%lld",A+i);
    //进行预处理
    for ( int i=1 ; i<=n ; i++ )//B[i]的值为 数组 A 中与 A[i] 相同的元素个数
        if ( A[i]==A[i-1] )//直接用已有的结果就行。A[1]>0=A[0] ，所以不用考虑 i=1 的情况
            B[i]=B[i-1];
        else//A[i]>A[i-1]，因此 j 从 i 开始循环。由于 A[n+1]=0 ，所以不用考虑陷入死循环
            for ( int j=i ; A[i]==A[j] ; j++,B[i]++ );
    //预处理完成，下面开始计算
    for ( int i=1 ; i<=n-2 ; i++ )
        for ( int j=i+1,k=n ; j<k ; j++ ){
            for ( ; k>j && A[i]+A[j]+A[k]>m ; k-- );
            if ( j<k && A[i]+A[j]+A[k]==m )
                if ( A[j]==A[k] )//值相同的时候，第三元可以从 k 一直取到 j+1 共 k-j 种
                    ans+=k-j;
                else//第三元可以取所有值为 A[k] 的元素，共 B[k] 种
                    ans+=B[k];
        }
    printf("%lld",ans);
```

```
    return 0;
}
```