A - 加权平均分计算

难度	考点
1	简单循环

题目分析

本题作为第四次上机的签到题，比较简单，主要考察了简单循环和数学运算。根据题干要求进行代码编写即可。

易错点提示

最后两个数相除的地方很容易写成整数相除，实际上应当转换为浮点数后再进行运算，才能得到正确的结果。

样例程序

```c
#include <stdio.h>
int main()
{
    int n, Sum1 = 0, Sum2 = 0, a, b; // Sum1 和 Sum2 分别作为分子和分母
    double score;

    scanf("%d", &n);
    while (n--) // n 次循环
    {
        scanf("%d%d", &a, &b);
        Sum1 = Sum1 + a * b;
        Sum2 = Sum2 + a;
    }
    score = (double)Sum1 / (double)Sum2; // 注意类型转换
    printf("%.2f", score);               // 保留 2 位小数

    return 0;
}
```

B - 复利

难度	考点
2	循环

题目分析

题目中的复利终值系数表的纵坐标是年份，横坐标是利率，并且都是整数。分析样例可知第一行是一年后的系数 $1.0100, 1.0200, 1.0300...$，第二行是第一行的平方，第三行是第一行的立方，依此类推。

因此本题用一个二重循环即可搞定。

解法 1

预处理最大范围的情况，将结果保存在二维数组中，之后读入数据并直接输出即可。

解法 2

采用 `math.h` 头文件中的 `pow()` 函数直接进行幂运算，例如 `pow(1.5, 2)` 会返回 1.5^2。

易错点提示

- 涉及到浮点数运算的情况下，首先推荐使用 `double` 类型运算，因为 `double` 类型的精度和可表示的范围都比 `float` 类型大很多。
- 再开数组的时候注意要留有余地，比如本题范围 30，有的同学就会只开 30，大概率就会 OE 或者 REG，推荐的数组大小——最大范围加十即可。

示例代码

解法 1

```cpp
#include <stdio.h>
#define maxn 100

int main()
{
    double a[maxn][maxn];
    int n, m, i, j;

    // 预处理，将结果保存在二维数组中
    for (j = 1; j <= 30; ++j)
    {
        double ans = 1;
        for (i = 1; i <= 30; ++i)
        {
            ans *= (1 + 0.01 * j);
            a[i][j] = ans;
        }
    }

    while (~scanf("%d%d", &n, &m))
```

```
    {
        // 直接输出即可
        for (i = 1; i <= n; ++i)
        {
            for (j = 1; j <= m; ++j)
            {
                printf("%.4lf ", a[i][j]);
            }
            printf("\n");
        }
    }

    return 0;
}
```

解法2

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

int main()
{
    int n, m;
    double a;

    while (~scanf("%d%d", &n, &m))
    {
        for (int i = 1; i <= n; ++i)
        {
            for (int j = 1; j <= m; ++j)
            {
                // math.h 头文件中的 pow 函数可以直接进行幂运算
                printf("%.4lf ", pow(1 + 0.01 * j, i));
            }
            printf("\n");
        }
    }

    return 0;
}
```

C - OO

难度	考点
2	条件控制

题目分析

先计算出圆心距、半径之差、半径之和，然后按照题目描述用 `if-else` 结构按优先级逐个判断即可。

易错点分析

多组数据情况下，需要视题目要求确定每组数据末尾要不要输出换行符，避免因为两组数据之间缺少换行符分隔而导致获得 WA。

示例程序

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

int main()
{
    int x1, x2, y1, y2, r1, r2;
    int d, add, sub;

    while (~scanf("%d%d%d%d%d%d", &x1, &y1, &r1, &x2, &y2, &r2))
    {
        d = (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2); // 圆心距
        add = (r1 + r2) * (r1 + r2); // 半径之和
        sub = (r1 - r2) * (r1 - r2); // 半径之差

        if (x1 == x2 && y1 == y2 && r1 == r2) // 重合
        {
            printf("fu zhi zhan tie bu xiang ma\n");
        }
        else if (r1 == r2) // 全等
        {
            printf("zong you ren yao yong jian pan ai ge qiao\n");
        }
        else if (x1 == x2 && y1 == y2) // 同心
        {
            printf("Ctrl+C/V tian xia di yi\n");
        }
        else if (sub > d) // 内含
        {
            printf("zhe ci ying gai dou hui fu zhi le ba\n");
        }
        else if (sub == d) // 内切
        {
            printf("oo0000000ooo0000oo0ooo000oo\n");
        }
        else if (sub < d && d < add) // 相交
        {
```

```
            printf("lIllIIlI11lIIIlIl1l1111\n");
        }
        else if (add == d) // 外切
        {
            printf("rrnnmnrmrnmrnrmnrmrnrm\n");
        }
        else if (add < d) // 相离
        {
            printf("qpgqopgqopgopqgpqggqpoogoo\n");
        }
    }

    return 0;
}
```

D - 聚类分析

难度	考点
3	二重循环与简单比较条件

题目分析

根据题目描述，进行二重循环并进行比较即可求解。

第一层循环遍历全部的待分类点，设遍历的下标变量为 i，第二层循环遍历全部的中心点，设遍历的下标为 j，则可求出当前待分类点 i 到中心点 j 的距离 d_{ij}。在每次第二层循环结束前对 d_{ij} 进行比较并求最大值，就可以求出 D_i，即 $D_i = \max\limits_{j=1}^{k} d_{ij}$。

对每一个点 i，可以用一个变量 `min_distance` 记录当前距离最小值，在循环 j 的时候不断更新它，循环结束后 `min_distance` 的值就是 D_i。为了方便起见，`min_distance` 的初值可以是一个很大的值（比如本题坐标很小，可以将距离设为可能最远距离的平方），或是点 i 与任意一个中心点 j 的距离，都可以让 `min_distance` 正确地被计算。

最后对 D_i 求和即可求解。

易错点提示

对于每个点 i，都需要在枚举 j 前重新对 `min_distance` 赋值，避免只有第一次循环获得了正确的 `min_distance`，而在第二次及之后的计算中忘记复原 `min_distance`，导致结果出错。

示例程序

```c
#include <stdio.h>

int main()
{
    int x[1100], y[1100], X[60], Y[60];
    int n, k, sum = 0;
    int i, j;
    int min_distance, temp_distance;

    scanf("%d%d", &n, &k);
    for (i = 0; i < n; i++)
    {
        scanf("%d%d", &x[i], &y[i]);
    }
    for (i = 0; i < k; i++)
    {
        scanf("%d%d", &X[i], &Y[i]);
    }

    for (i = 0; i < n; i++)
    {
        // 每次循环开头重置 min_distance
        // 这里 0x7fffffff 是 int_max
        min_distance = 0x7fffffff;
```

```
        for (j = 0; j < k; j++)
        {
            temp_distance = (x[i] - X[j]) * (x[i] - X[j]) + (y[i] - Y[j]) *
(y[i] - Y[j]);
            // 当前值比最大值还大时，更新结果
            if (min_distance > temp_distance)
            {
                min_distance = temp_distance;
            }
        }
        sum += min_distance;
    }

    printf("%d", sum);

    return 0;
}
```

E - Halloweeeeeeeeeeeeeeeeeen

难度	考点
3	循环控制，条件控制

题目分析

做法 1

将 n 个数读入之后，枚举每个区间 $[i, j]$ $(1 \leq i \leq j \leq n)$，并判断区间内数字是否全为 "1"，若符合，则更新答案和起始位置。（此做法仅适用于 n 较小的情况）

做法 2

一边读入一遍判断，以 len 记录当前连续 "1" 的长度，当读到 "0" 则更新答案和起始位置，并将 len 归零。

样例程序 1

```c
#include <stdio.h>

int main()
{
    int t, n, i, j, k;
    int a[233];
    int ans, p, tag;

    scanf("%d", &t);
    while (t--)
    {
        scanf("%d", &n);
        for (i = 1; i <= n; i++)
            scanf("%d", &a[i]);

        // ans 为最长长度，p 为第一次出现的位置
        ans = 1, p = 1;

        for (i = 1; i <= n; i++)
        {
            for (j = n; j >= i; j--)
            {
                tag = 1;
                // 检查区间为 [i, j] 是否有 0
                // 有 0 则 tag 被改为 0
                for (k = i; k <= j; k++)
                {
                    if (a[k] == 0)
                        tag = 0;
                }
                // 区间全 1
                if (tag == 1)
                {
```

```
                    if (j - i + 1 > ans)
                    {
                        ans = j - i + 1;
                        p = i;
                    }
                }
            }
        }
        printf("%d %d\n", ans, p);
    }

    return 0;
}
```

样例程序 2

```c
#include <stdio.h>

int main()
{
    int n, t, tmp, i;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%d", &n);
        int ans = 1, p = 1, len = 0;
        for (i = 1; i <= n; i++)
        {
            scanf("%d", &tmp);
            if (tmp == 1)
                len++;
            else
            {
                if (len > ans)
                {
                    ans = len;
                    p = i - len;
                }
                len = 0;
            }
        }
        if (len > ans)
        {
            ans = len;
            p = n - len + 1;
        }
        // 注意若读入的最后一位为 1，len 不为 0，需多进行一次判断
        printf("%d %d\n", ans, p);
    }
}
```

F - 方形「方形造形術」

难度	考点
4	多重循环

题目分析

本题有很多解法，核心思路是“找规律”，即找到输出与行数之间的关系，并用循环输出。

以下提供一些思路参考：

- 图案是很美观的**对称**图形；
- 图案输出分为两种，背景和字母，其中字母是按顺序输出的。

以下提供两个示例程序，分别是：

1. 思路非常简单、解法**极其暴力**，但是十分冗长的解法。适合头脑混乱时无脑解题用；
2. 优雅但是不那么容易想到的解法。适合头脑清晰时使用。

PS：笔者认为本题的图形还挺美的，大家可以用 `'|'` `'\'` `'/'` `'>'` `')'` `'*'` `'+'` `'.'` `'^'` 等各种字符做背景看看效果。

示例程序 1

主要思路：

将图案分拆为上下两部分，只对上部分进行思考，下部分则直接复制上部分代码改变循环顺序输出；将图案的每一行分为三部分，分别是左背景、字母、右背景，再分别进行循环输出。

对于第 i 行，其字母数量为 $2i+1$，而左右背景字符数分别为 $n-i-1$。而字母又是回文结构，同样可以先输出前半部分，再将后半部分反向输出。

具体代码如下：

```c
#include<stdio.h>

int main()
{
    int bg, n;  // bg: background
    int i, j, k, l, fast, close;
    char char_out;
    char a[100];

    scanf("%c %d", &bg, &n);
    for (i = 0; i < n; i++) {   // 上半部分（含中线），循环从 i 递增至 n - 1
        for (j = 0; j < n - i - 1; j++) {   // 输出左背景
            putchar(bg);
        }
        // 输出字母
        for (j = 0; j <= i; j++) {  // 字母前半部分，含中心
            char_out = 'A' + j;
            printf("%c", char_out);
        }
```

```
        for (j = i - 1; j >= 0; j--) {  // 字母后半部分，注意对比循环条件与之前的不同
            char_out = 'A' + j;
            printf("%c", char_out);
        }

        for (j = 0; j < n - i - 1; j++) {   // 输出右背景
            putchar(bg);
        }
        printf("\n");
    }

    for (i = n - 2; i >= 0; i--) {  // 下半部分（不含中线），循环从 n - 2 递减至 0
        for (j = 0; j < n - i - 1; j++) {
            putchar(bg);
        }

        for (j = 0; j <= i; j++) {  //字母前半部分，含中心
            char_out = 'A' + j;
            printf("%c", char_out);
        }
        for (j = i - 1; j >= 0; j--) {
            char_out = 'A' + j;
            printf("%c", char_out);
        }

        for (j = 0; j < n - i - 1; j++) {
            putchar(bg);
        }
        printf("\n");
    }

    return 0;
}
```

事实上，这个思路完全不需要写得如此啰嗦，完全可以只写一个双重循环，然后在循环体内用三目运算或条件语句判断输出。

所以同学们可以借鉴一下这个思路，试着写写更优雅的代码。

示例程序 2

以图案中心为原点建立坐标系，可以知道，输出字符是字母的充要条件是：$|x| + |y| < n$

这种写法就是以此为核心思路编写的。

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

int main()
{
    char bg;
    int n;
    int i, j, d;

    scanf("%c %d", &bg, &n);

```

```
    for(i = -(n - 1); i <= n - 1; i++)
    {
        for(j = -(n - 1); j <= n - 1; j++)
        {
            d = abs(i) + abs(j);
            if (d < n) {
                printf("%c", 'A' + n - 1 - d);
            } else {
                printf("%c", bg);
            }
        }
        printf("\n");
    }

    return 0;
}
```

G - Cell-3

难度	考点
5	递推，数组，取模的性质

解法 1

采用数组递推。~~每~~一天的细胞数量可以分成两部分：前一天的细胞数量 + 这一天新生的细胞数量。设 f_i 为第 i 天细胞的数量 $(i \geq 0)$，前一天的细胞数量为 f_{i-1} $(i \geq 1)$，而这一天新生的细胞数量为**当天具备分裂能力的细胞**的数目乘以 2，而根据题目条件，出生第 3 天(含)到第 6 天(不含)之间的细胞是具有分裂能力的，其数目为 $f_{i-3} - f_{i-6}$ $(i \geq 6)$。所以根据上述分析可得递推公式 $f_i = f_{i-1} + 2 \cdot (f_{i-3} - f_{i-6})$。当前天数不足 3 天或者不足 6 天时，$f_{i-3}$ 或 f_{i-6} 相应地为零。

这个数组的值在随着天数增大时数量级会迅速升高，如果等到全部计算完结果后再取模则运算过程中就会溢出，所以根据取模的性质需要在每次计算加法或乘法时取模。在编程计算中，数组元素 `f[i]` 真实表示的含义为上述细胞数量 f_i 对给定模数取模后的结果。根据提示 3 所给的不等式，在给定的模数下，两项相加一定不会溢出 `int` 的范围，而三项及以上的元素相加是可能会溢出的，因此我们需要做到对**每一步**加法运算都取一次模（乘以 2 视同两项相加），这样才可以避免运算过程中的溢出（~~当然，欢迎 `long long` 水过~~）。另一方面，由于递推公式中含有减法，并且 `f[i]` 是一个取模后的结果，所以前后项的大小关系并不能保证，因此我们需要手动防止减法得到负数。根据取模的加法性质，$a \equiv (a + p) \bmod p$，所以若 `m % p` 中的 `m` 不能保证为正数时，可写成 `(m + p) % p` 来保证结果一定是正数。

最后，注意模数是 $10^9 + 9$ 而不是 $10^9 + 7$，以及题目要求输出的是第 $3n$ 天的结果（不是第 n 天），所以注意数组长度也要相应开为 n 最大范围的 3 倍。

示例程序 1

```c
#include <stdio.h>

#define MOD 1000000009 // 采用 #define 简化代码
#define MAXN 1000005

int f[MAXN * 3]; // 注意数组长度

int main()
{
    int n, i;

    scanf("%d", &n);
    n *= 3; // 注意乘以3
    f[0] = 1;
    for (i = 1; i <= n; i++)
    {
        // 由于每一步都要取模，所以可以分多步来计算，防止中间忘记每一步都取模或者式子太长
        f[i] = f[i - 1];
        if (i >= 3)
            f[i] = (f[i] + (2 * f[i - 3]) % MOD) % MOD;
        if (i >= 6)
            f[i] = (f[i] - (2 * f[i - 6]) % MOD) % MOD;
        f[i] = (f[i] + MOD) % MOD; // 规避负数
    }
```

```
    printf("%d", f[n]);

    return 0;
}
```

解法 2

可以采用一个长度为 7 的数组，根据细胞的"年龄"（即出生后的天数）对不同状态的细胞进行分类。

数组 `day[m]` 表示当循环到第 i 天时各个年龄的细胞数量，其中 `day[0]` 为当天新出生的细胞数量，`day[6]` 为出生**大于等于** 6 天的细胞数量（即过了可分裂期的细胞数量）。在这种方法下，每天具有分裂能力的细胞数量即为 `day[3] + day[4] + day[5]`（在写代码时务必注意每加一次就要取模一次）。每天更新该数组，最后当循环完 $3n$ 天时对 `day` 数组求和。注意求和时仍然要注意每做一次加法就要取一次模，这里可以采用循环来求和，防止溢出。

示例程序 2

```
#include <stdio.h>

#define MOD 1000000009
#define N 1000005

int day[7];

int main()
{
    int n, d, i, sum = 0;

    scanf("%d", &n);
    n = n * 3;
    day[0] = 1;
    for (d = 1; d <= n; d++)
    {
        day[6] += day[5];
        day[6] %= MOD;
        for (i = 5; i >= 1; i--)
        {
            day[i] = day[i - 1];
        }
        day[0] = (((2 * day[3]) % MOD + (2 * day[4]) % MOD) % MOD + (2 * day[5]) % MOD) % MOD;
    }
    for (i = 0; i <= 6; i++)
    {
        sum = (sum + day[i]) % MOD;
    }
    printf("%d", sum);

    return 0;
}
```

H - 三角形缩小术

难度	考点
5	简单循环

题目分析

从 x 缩小到 y 的策略比较难想，故将问题倒转：将三角形从 y 逐渐放大。逐渐放大可以暴力贪心，每次把最短的边修改为 $\min(x, \text{另两条边之和} - 1)$ 变为最长边。直到三条边都为 x 时的操作次数即为最小的操作数。容易证明，倒序的放大操作就是所求缩小操作。

示例程序

```c
#include <stdio.h>

int main()
{
    int x, y, t;
    int a[3], now;

    scanf("%d", &t);
    while (t--)
    {
        scanf("%d%d", &x, &y);
        a[0] = a[1] = a[2] = y;
        now = 0;
        while (1)
        {
            if (a[0] >= x && a[1] >= x && a[2] >= x) break;
            now++;
            if (now % 3 == 1)
                a[0] = a[1] + a[2] - 1;
            else if (now % 3 == 2)
                a[1] = a[0] + a[2] - 1;
            else
                a[2] = a[0] + a[1] - 1;
        }
        printf("%d\n", now);
    }

    return 0;
}
```

I - 欧拉和蔡勒

难度	考点
6	Zeller 公式的应用，循环结构

题目分析

在使用蔡勒公式计算星期数时，要注意以下两点：

- 一定要对 1582 年 10 月 4 日或之前的日期和 1582 年 10 月 15 日或之后的日期使用不同的公式。
- 计算出的 W 可能是负数，可以使用 $W = (W + 7) \bmod 7$ 的方式将其变为正数。

注意到 $i \mid j$ 当且仅当 i 是 j 的因子，可以使用 素数判定 中的试除法来找到 1 到 n 每个数的因子，但这样做的时间复杂度为 $O(n\sqrt{n})$，无法通过本题，需要另辟蹊径。

通过对欧拉常数的定义式进行变形可以得到

$$\gamma = \lim_{n\to\infty}[(\sum_{i=1}^{n}\frac{1}{i}) - \ln(n)] \approx 0.5772$$

$$\Rightarrow \sum_{i=1}^{n}\frac{1}{i} \sim \ln(n)\ (n \to \infty)$$

$$\Rightarrow \sum_{i=1}^{n}\frac{n}{i} \sim n\ln(n)\ (n \to \infty)$$

因此，当 n 较大时，我们可以认为 $\sum\limits_{i=1}^{n}\frac{n}{i}$ 近似和 $n\ln(n)$ 相等。考虑 $1, 2, \cdots, n$ 对于它的倍数的贡献，例如 1 对 $1, 2, \cdots, n$ 有贡献，2 对 $2, 4, \cdots, 2\left\lfloor\frac{n}{2}\right\rfloor$ 有贡献，这样总计算次数为

$$\left\lfloor\frac{n}{1}\right\rfloor + \left\lfloor\frac{n}{2}\right\rfloor + \cdots + \left\lfloor\frac{n}{n}\right\rfloor \le \sum_{i=1}^{n}\frac{n}{i}$$

时间复杂度为 $O(n\log n)$，足够通过本题。

示例程序

```c
#include <stdio.h>

const int p = 998244353;

int f[1000086];

int main()
{
    int n, y, m, d, c, w, i, j, ans = 0;

    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        f[i] = 1;

    for (i = 1; i <= n; i++)
    {
        scanf("%d%d%d", &y, &m, &d);
```

```
        // 根据蔡勒公式计算星期 w
        if (m < 3)
            y--, m += 12;
        c = y / 100;
        if (y < 1582 || (y == 1582 && m < 10) || (y == 1582 && m == 10 && d
<= 4))
        {
            w = (y % 100) + (y % 100) / 4 + c / 4 - 2 * c + 13 * (m + 1) / 5
+ d + 2;
        }
        else
        {
            w = (y % 100) + (y % 100) / 4 + c / 4 - 2 * c + 13 * (m + 1) / 5
+ d - 1;
        }
        w = ((w % 7) + 7) % 7;
        if (w == 0) w = 7;

        // 计算 i 对其倍数的贡献
        // 注意运算过程中可能会超出 long long，因此要边做边取模
        for (j = 1; i * j <= n; j++)
            f[i * j] = 1LL * f[i * j] * w % p;
    }

    // 注意运算过程中可能会超出 long long，因此要边做边取模
    for (i = 1; i <= n; i++)
        ans = (ans + f[i]) % p;
    printf("%d", ans);

    return 0;
}
```

J - 德州扑克

难度	考点
6	循环结构，判断结构，综合控制流

题目分析

本题中，Alice 和 Bob 的手牌实际上本质相同，因此如果我们能将两人的手牌分别转换成能直接比较大小的数字保存下来，那么数字孰大孰小就对应了手牌孰大孰小。

因此，我们可以记录一个 `type` 和 `value`，分别表示手牌的牌型是 `type`，以及它在同牌型下的相对大小是 `value`。对于 `type`，我们只要根据题目的要求判断手牌属于 7 个牌型中的何种，依次为其赋值即可。对于 `value`，我们发现同牌型内的比较全都可以转换为：先比较某张牌的大小，若相同则比较下张牌的大小，若相同则继续比较……我们发现这样的比较方法很像比较两个数字：先比较第一位的大小，若相同则比较下一位的大小，因此我们可以用一个足够大的进制的每一位去表示这个牌型对应的每张牌，这样数的大小就直接对应一个牌型大小了。

以散牌为例，不妨设手牌为 $2, 5, 7, 11, 13$ 与 $3, 5, 7, 11, 13$，则其在十六进制下可以表示为 db752 与 db753，即十进制的 898898 与 898899，因此可以直接比较出前者牌型比后者牌型小。

具体的实现方法可以参考示例程序。

示例程序

```c
#include <stdio.h>

// 采用 16 进制表示手牌，故 BASE = 16
// POWX 表示 BASE 的 X 次幂
#define BASE (16)
#define POW1 (BASE)
#define POW2 (POW1 * BASE)
#define POW3 (POW2 * BASE)
#define POW4 (POW3 * BASE)

int main()
{
    int type[2], value[2];
    int n;
    int i;
    int op, a, b, c, d, e;

    scanf("%d", &n);
    for (i = 0; i < 2 * n; i++)
    {
        scanf("%d%d%d%d%d", &a, &b, &c, &d, &e);
        op = i & 1; // 当前的操作方是谁

        // 判断手牌类型
        if (a == d || b == e)
            type[op] = 1; // 四条
        else if ((a == c && d == e) || (a == b && c == e))
            type[op] = 2; // 葫芦
```

```
        else if (a == b - 1 && b == c - 1 && c == d - 1 && d == e - 1)
            type[op] = 3; // 顺子
        else if (a == c || b == d || c == e)
            type[op] = 4; // 三条
        else if ((a == b && c == d) || (a == b && d == e) || (b == c && d == e))
            type[op] = 5; // 两对
        else if (a == b || b == c || c == d || d == e)
            type[op] = 6; // 一对
        else
            type[op] = 7; // 散牌

        // 计算分数
        switch (type[op])
        {
        case 1: // 四条
            if (a == d) value[op] = POW1 * a + e;
            if (b == e) value[op] = POW1 * e + a;
            break;
        case 2: // 葫芦
            if (a == c) value[op] = POW1 * a + e;
            if (c == e) value[op] = POW1 * e + a;
            break;
        case 3: // 顺子
            value[op] = e;
            break;
        case 4: // 三条
            if (a == c) value[op] = POW2 * c + POW1 * e + d;
            if (b == d) value[op] = POW2 * c + POW1 * e + a;
            if (c == e) value[op] = POW2 * c + POW1 * b + a;
            break;
        case 5: // 两对
            if (a == b && c == d) value[op] = POW2 * d + POW1 * b + e;
            if (a == b && d == e) value[op] = POW2 * e + POW1 * b + c;
            if (b == c && d == e) value[op] = POW2 * e + POW1 * c + a;
            break;
        case 6: // 一对
            if (a == b) value[op] = POW3 * b + POW2 * e + POW1 * d + c;
            if (b == c) value[op] = POW3 * c + POW2 * e + POW1 * d + a;
            if (c == d) value[op] = POW3 * d + POW2 * e + POW1 * b + a;
            if (d == e) value[op] = POW3 * e + POW2 * c + POW1 * b + a;
            break;
        case 7: // 散牌
            value[op] = POW4 * e + POW3 * d + POW2 * c + POW1 * b + a;
        }

        if (op == 1)
        {
            // 先比较牌型大小
            if (type[0] != type[1])
            {
                printf(type[0] < type[1] ? "Alice\n" : "Bob\n");
            }
            else
            {
                // 再比较同牌型情况下的手牌大小
                if (value[0] != value[1])
                    printf(value[0] > value[1] ? "Alice\n" : "Bob\n");
```

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

    return 0;
}
```