C4 - Solution

A 免修考试结果

难度	考点
1	分支结构

题目分析

利用分支结构分类讨论，按照题目要求输出即可。

示例代码

```c
#include <stdio.h>

int main()
{
    int year, score;
    while(scanf("%d%d", &year, &score) != EOF)
    {
        if(year == 2023 && score >= 85)
        {
            printf("%d\n", score);
            puts("You can choose whether to take the course or not.");
        }
        else if(year != 2023 && score >= 60)
        {
            printf("%d\n", score);
        }
        else
        {
            puts("You have to take the course.");
        }
    }
    return 0;
}
```

B AMI编码

难度	考点
1	分支结构

题目解析

可以利用一个 `flag` 变量来记录 `1` 的符号。每读入一个字符，判断它是 `0`，`1` 或其它字符。如果是 `0`，那么将它原样输出。如果是 `1`，那么输出 `flag`，然后把 `flag` 变成自己的相反数。如果是 `EOF`，直接结束程序。

示例代码

```
#include <stdio.h>

int main()
{
    char c;
    int flag = 1;
    while ((c = getchar()) != EOF)
    {
        if (c == '0')
            printf("0 ");
        else // c == '1'
        {
            if (flag > 0)
                printf("+1 ");
            else
                printf("-1 ");
            flag *= -1;
        }
    }
    return 0;
}
```

C 绩效评定

难度	考点
2	判断

题目分析

本题实际上是含特判的多分支题目。我们注意到所给业绩得分按照降序排列，因此本题可以通过**如果当前编辑的业绩得分与上一位相同，则（进入特判）该编辑的评定等级继承上一位编辑的等级**来先处理特判。后续的多分支情况注意使用 `else` 使各情况互斥即可。

判断名次的区间最好使用整型变量判断，不会有误差。如 $r \leq n \cdot 10\%$ 等价于 $10 \times r \leq n$ 。

示例代码

```
#include <stdio.h>
int main()
{
    int n;
    int grade, c = 'A', lastgrade = -1; //业绩得分，等级，上一个业绩得分
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        scanf("%d", &grade);
        if (grade == lastgrade) //与上一位编辑业绩得分相同，等级也相同，直接输出。
        {
            printf("%c ", c);
            continue;
        }
        if (i * 10 <= n)
            c = 'A';
        else if (i * 4 <= n)
```

```
            c = 'B';
        else if (i * 10 <= n * 6)
            c = 'C';
        else if (i * 10 <= n * 9)
            c = 'D';
        else
            c = 'E';
        printf("%c ", c);
        lastgrade = grade; //更新上一位编辑的业绩
    }
    return 0;
}
```

D 抢麦大作战

难度	考点
2	模拟

题目分析

可以使用数组存储当前每种普通小食的数量。每打出一张牌检查是否有普通小食的数量等于 9，如果有那么输出相应内容并将其数量清零。

注意到打出一张牌前必然没有普通小食的数量等于 9，数量变为 9 的普通小食只可能是当前打出的牌含有的小食。

示例代码

```
#include <stdio.h>
int main()
{
    int i, a, b, c[6] = {0};
    for (int i = 1; scanf("%d%d", &a, &b) != EOF; i++)
    {
        c[a] += b;
        if (c[a] == 9)
        {
            printf("%d %d\n", i, a);
            c[a] = 0;
        }
    }
    return 0;
}
```

E 真实命中率

难度	考点
2~3	统计，数组，循环分支结构，浮点运算

题目分析

建立三个数组分别存储每位球员的全场得分，是全场出手次数（两分球或三分球，不算罚球），罚球出手次数。

每次读入一个数据，对相应的球员的数据进行修改，最后按照公式计算每个球员的真实命中率即可。

当且仅当 FTA 和 FTA 均为 0 时，没有该球员的数据记录。

注意整型进行除法运算时需要转为 `double` 类型计算，否则会变为整除（向下取整），本题示例代码中通过隐式强制转换保证了结果的正确。

不要忘了如何用 `printf` 输出 `%`。

示例代码

```
#include <stdio.h>

int main()
{
    int m, i, x, y, PTS[15] = {0}, FGA[15] = {0}, FTA[15] = {0};
    scanf("%d", &m);
    while(scanf("%d%d%d", &i, &x, &y) != EOF)
    {
        PTS[i] += x * y; // if(y == 1) PTS[i] += x;
        x == 1 ? FTA[i]++ : FGA[i]++; // 使用if/else也可以
    }
    for(i = 1; i <= m; i++)
    {
        if(FGA[i] || FTA[i]) // 有球员i的数据记录
            printf("%.2f%%\n", PTS[i] / (FGA[i] + 0.44 * FTA[i]) / 2 * 100);
        else // 没有球员i的数据记录
            puts("NULL");
    }
    return 0;
}
```

F 连分数与辗转相除

难度	考点
3	函数、递归、辗转相除法

题目分析

我们发现，设 $\frac{p}{q}=[a_0,a_1,\cdots,a_k]$， a_0 等于 $\frac{p}{q}$ 向下取整，而 $[a_1,a_2,\cdots,a_k]$ 正是 $\frac{q}{p-a_0q}=\frac{q}{p \mod q}$ 的连分数表示，可见该过程与辗转相除法几乎完全相同。仿照辗转相除法，将每次除法的商输出即可。

示例代码

```
#include <stdio.h>
int main()
{
    int p, q;
    scanf("%d%d", &p, &q);
    while(q != 0)
    {
        printf("%d ", p/q); //输出a_0
        int t = p % q;
```

```
        p = q; //更新p为q
        q = t; //更新q为p%q
    }
    return 0;
}
```

补充

在下次课学习了函数之后，也可以利用递归函数来实现辗转相除法。同学们可以在下次课上完之后重新回顾一下本题。示例代码如下：

```
#include <stdio.h>
void f(int p, int q)
{
    if(q == 0) return;
    printf("%d ", p / q); //输出a_0
    f(q, p % q); //若p%q不为0，继续递归计算a1,...,ak
}
int main()
{
    int p, q;
    scanf("%d%d", &p, &q);
    f(p, q);
    return 0;
}
```

G 分数四则运算

难度	考点
4	循环分支结构，最大公约数

题目分析

按照题意一步一步模拟，首先按输入格式进行读入，然后根据 op 的值进入对应的分支，得到结果后对分子分母进行约分，最后注意一下符号和是否为整数，按要求输出答案即可。

注意由于运算过程中可能会超过 `int` 范围，需要使用 `long long` 进行运算。

示例代码

```
#include <stdio.h>

int main()
{
    int op;
    long long a, b, c, d, p, q;
    while(~scanf("%lld/%lld %lld/%lld %d", &a, &b, &c, &d, &op))
    {
        switch(op)
        {
            case 1:
                p = a * d + b * c;
                q = b * d;
                break;
            case 2:
                p = a * d - b * c;
```

```
                q = b * d;
                break;
            case 3:
                p = a * c;
                q = b * d;
                break;
            case 4:
                p = a * d;
                q = b * c;
                break;
        }

        // 求最大公约数
        long long x = p, y = q;
        while(y != 0)
        {
            long long t = x % y;
            x = y;
            y = t;
        }
        // 此时x即为p和q的最大公约数

        // 约分
        p /= x;
        q /= x;

        // 保证 q > 0
        if(q < 0)
        {
            p = -p;
            q = -q;
        }

        printf("%lld", p);
        if(q != 1) // 不是整数，输出分母
            printf("/%lld", q);

        printf("\n");
    }
    return 0;
}
```

H 想和De心有灵犀

难度	考点
5	求余，构造

题目分析

我们可以很轻松想到考虑相邻两数的和，即两个数的和尽量为 p 的倍数

由于余数域为 $0 \sim p-1$ ，所以我们考虑对于余数为0的数，只有与同样余数为0的数相邻，才对结果有贡献；对于余数非0的数，若为 k ，那么只能在与余数为 $p-k$ 的数相邻时才能对答案有贡献

所以我们要答案尽可能大，只需要将可以满足条件的数尽量相邻即可

具体过程：循环余数 $i: 0 \sim \lfloor \frac{p}{2} \rfloor$ ，先放 $p \times 0 + i$，接下来放入余数为 $i = (p-i)\%p$ 的数 $p \times 0 + i$ ，同上，对 i 进行更新，放入 $p \times 1 + i \cdots$ 直至 $p \times m + i > n$ 时停止当次循环，进行下一个余数的循环

示例代码

```
#include <stdio.h>

#define maxn 100005

int test, n, p, ans, res, now;
int cnt[maxn];

int main()
{
    scanf("%d", &test);
    while(test--)
    {
        memset(cnt, 0, sizeof(cnt));
        cnt[0] = 1;
        scanf("%d %d", &n, &p);
        if(p == 1)
        {
            for(int i = 1; i <= n; i++)
                printf("%d ", i);
            printf("\n");
        }
        else
        {
            int lim = p / 2;
            for(int i = 0; i <= lim; i++)
            {
                now = i;
                if(now > n) break;
                res = (cnt[now]++) * p + now;
                now = (p - now) % p;
                while(res <= n)
                {
                    printf("%d ", res);
                    if(now > n) break;
                    res = (cnt[now]++) * p + now;
                    now = (p - now) % p;
                }
            }
            printf("\n");
        }
    }
    return 0;
}
```

I 最大向量价值

难度	考点
6~7	状态压缩

题目分析

假设我们已经选取了 k 个 m 维向量 $\overrightarrow{a[1]}, \overrightarrow{a[2]}, \ldots, \overrightarrow{a[k]}$，得到了一个和向量 $\vec{v} = (v_1, v_2, \cdots, v_m) = (a[1]_1 + a[2]_1 + \ldots + a[k]_1, a[1]_2 + a[2]_2 + \ldots + a[k]_2, \ldots, a[1]_m + a[2]_m + \ldots + a[k]_m)$ ；对应于数列 v ，我们构造一个符号数列 b ，如下:

$$b_i = \begin{cases} 1 & , v_i \geq 0 \\ -1 & , v_i < 0 \end{cases}$$

则 $\vec{v}$ 对应的向量价值为 $value = \sum_{i=1}^{m} b_i v_i = \sum_{i=1}^{m} b_i (a[1]_i + a[2]_i + \ldots + a[k]_i) = \sum_{j=1}^{k} (b_1 a[j]_1 + b_2 a[j]_2 + \ldots + b_m a[j]_m)$ 。

简单来说，如果我们确定了最后求出的和向量各维度符号为 $(b_1, b_2, \ldots, b_m)$ ，那么被选中的某个向量 $\vec{a} = (a_1, a_2, \ldots, a_m)$ 对于这个和向量向量价值的贡献为 $cont = b_1 a_1 + b_2 a_2 + \ldots + b_m a_m$ ；如果 $cont > 0$ ，那么就说明这个被选中的向量对和向量做了正贡献，使其向量价值增大；如果 $cont < 0$ ，那么就说明这个被选中的向量对和向量做了负贡献，使其向量价值减小；如果 $cont = 0$ ，那么就说明无论选不选这个向量都不会影响最后的向量价值。

对于某个确定的维度 m ，选取若干向量求得的和向量的符号序列 $(b_1, b_2, \ldots, b_m)$ 有多少种可能呢？很显然，每一位都可以取到 -1 或 1 ，故最后的符号序列一共有 2^m 种可能。对于某确定的符号序列，符合这个符号序列的和向量的**可能的**最大向量价值是多少呢？我们可以遍历向量数组，用这个符号序列计算每个向量对于这种符号序列的和向量的贡献值，如果贡献值大于 0 ，那么就把它加入到和向量中。于是，我们得到了所有 2^m 情况对应的**可能**的最大值，然后我们就可以遍历 2^m 个情况和向量的最大值中选出一个最大的，作为选取若干向量组成和向量的最大向量价值。

这里需要特别说明的是，**每种情况求出的可能的最大向量价值是不一定取的到的，但是所有情况里最大的可能的向量价值是一定取的到的。**例如只有一个二维向量 $(-3, 7)$ 对于和向量符号情况 $(1, 1)$ 和 $(-1, 1)$ 都是有正贡献的，但是和向量是根本不可能取到 $(1, 1)$ 的，但是**一定能取到所有情况中能得到最大向量价值的情况**。这是因为 $|a| + |b| = max(a, -a) + max(b, -b) = max(a + b, a - b, -a + b, -a - b)$ ，取最大值时 a 和 b 前的符号与 a 和 b 的符号相对应，对于更多项的公式也是类似的。故和向量各维度绝对值之和取最大值时，各维度的符号一定与该符号序列状态的各维度（也就是公式中每个项前面的正负符号）相对应。

最后考虑具体实现，实际上，我们用一个正整数就能表示和向量的符号序列，这里与 `E3-J` 有一点相似之处。比如知道所有向量维度是 3，那么就可以用 6 ，即 110 表示符号序列 $(-1, -1, 1)$ ；用 1 ，即 001 表示符号序列 $(1, 1, -1)$ 等。我们用数组元素 val_i 表示对应符号序列为 i 的和向量的向量价值。每读入一个向量，我们就看看它对所有 2^m 个状态是否有正贡献，如果有正贡献，就把对应的贡献值加到 val_i 上。处理完所有向量后，最大的 val 值就是选取若干向量可能得到的最大的向量价值。

因为根据数据范围最大的值可能超过 int，所以用 long long 类型保存 val 数组。

示例代码

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

long long val[1 << 8];  // 向量最大维度 8 维，最多 1<<8 种状态
int vec[10];

int main()
{
    int n,m;
    scanf("%d%d",&m,&n);
    int sta = 1 << m;   // 状态数

    // 处理每个向量
    for(int i = 1;i <= n;i++)
    {
        // 读入某向量的全部 m 个维度，这里会覆盖上一次读入的数据
        for(int j = 1;j <= m;j++)
```

```
        scanf("%d",&vec[j]);

        // 依次遍历每一种可能的符号序列状态
        for(int j = 0;j < sta;j++) {
            // 计算对该状态的贡献
            int con = 0;
            for(int k = 1;k <= m;k++) {
                // 从状态里取出符号
                int op;
                if(((j >> (k-1)) & 1) == 0) {
                    op = 1;
                } else {
                    op = -1;
                }
                con += op * vec[k];
            }
            if(con > 0) val[j] += con; // 正贡献就加入
        }
    }

    // 开始找所有状态里的最大值
    long long maxVal = -1;
    for(int i = 0;i < sta;i++){
        if(val[i] > maxVal) {
            maxVal = val[i];
        }
    }

    printf("%lld\n",maxVal);
    return 0;
}
```

J 寻找进制

难度	考点
6~7	数论

题目分析

这个问题等价于寻找如下方程大于 $\max\{a_k, b_k, c_k\}$ 的最小正整数解：

$$\left(a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}\right)\left(b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}\right) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$$

也就是 $d_0 + d_1 x + \cdots + d_{n-1} x^{2n-1} = 0$，其中 $d_k = \left(\sum_{i+j=k} a_k b_k\right) - c_k$ 。

如果 $d_0 = d_1 = \cdots = d_{n-1} = 0$，那么大于 $\max\{a_k, b_k, c_k\}$ 的最小正整数解显然是 $\max\{a_k, b_k, c_k\} + 1$ 。

如果 $d_0 = d_1 = \cdots = d_{t-1} = 0$ 而 $d_t \neq 0$，那么 $d_t = -x\left(d_{t+1} + d_{t+2} x + \cdots + d_{n-1} x^{n-t-2}\right)$，这意味着方程的正整数解 x 一定是 d_t 的因子。

注意到不大于 10^{14} 的正整数的因子的数量不大于 17280，如果 x 是 d_t 的因数且 $x > \sqrt{d_t}$，那么 $\dfrac{d_t}{x}$ 是 d_t 的因数且 $x < \sqrt{d_t}$，这意味着我们只需要遍历不大于 $\sqrt{d_t}$ 的正整数就能得知 d_t 的全部因数。

为了判断某个 x 是否满足条件，我们需要注意到 $d_0 + d_1 x + \cdots + d_{n-1} x^{n-1} = 0$ 当且仅当 $d'_0 + d'_1 x + \cdots + d'_{n-2} x^{n-2} = 0$，其中 $d'_0 = d_1 + \dfrac{d_0}{x}, d'_{k+1} = d_k$ 。

示例代码

```c
#include <stdio.h>

int n;
long long int a[10000], b[10000], c[10000], d[20000];

int main() {
    scanf("%d", &n);
    long long int m = 0;
    for (int i = 0; i < n; i++) {
        scanf("%lld", &a[i]);
        if (a[i] > m) {
            m = a[i];
        }
    }
    for (int i = 0; i < n; i++) {
        scanf("%lld", &b[i]);
        if (b[i] > m) {
            m = b[i];
        }
    }
    for (int i = 0; i < n; i++) {
        scanf("%lld", &c[i]);
        if (c[i] > m) {
            m = c[i];
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            d[i + j] += a[i] * b[j];
        }
        d[i] -= c[i];
    }

    int t;
    for (t = 0; t < n * 2 && d[t] == 0; t++);
    if (t == n * 2) {
        printf("%lld", m + 1);
        return 0;
    } else {
        long long int val = d[t] < 0 ? -d[t] : d[t];
        long long int x;
        for(x = m + 1; x * x < val; x++) {
            if (val % x == 0) {
                long long int z = x, y = 0;
                for (int i = 0; i < n * 2; i++) {
                    y = d[i] + y / z;
                    if (y % z != 0) {
                        break;
                    }
                }
                if (y == 0) {
                    printf("%lld", z);
                    return 0;
                }
            }
        }
        if (x > val / (m + 1)) {
            x = val / (m + 1);
        }
```

```
        for (; x > 0; x--) {
            if (val % x == 0) {
                long long int z = val / x, y = 0;
                for (int i = 0; i < n * 2; i++) {
                    y = d[i] + y / z;
                    if (y % z != 0) {
                        break;
                    }
                }
                if (y == 0) {
                    printf("%lld", z);
                    return 0;
                }
            }
        }
        printf("error");
        return 0;
    }
}
```

- End -