CPL 第三次编程练习 3-for-a-while 题解

教学周历中本周课程的知识点为：

> 更多for循环案例、while与do while语句、break和continue语句。

教材章节：

> 6.1、6.2、6.4、6.5

Flip

> 本题知识点：
> 整数输入，整数输出，标记数组使用，for 循环，双重循环，**找规律** or 简单数学分析。

对于初学者，如果拿到题目没有思路，不妨先模拟一下题目的过程。

```
// 核心代码部分 - 模拟
int flag[2005];
memset(flag, 0, sizeof(flag));// memset的正确用法:(
for(int i = 1; i ⩽ n; i++){
    for(int j = 1; j ⩽ i; j++){
        if(i % j == 0){
            flag[i] ^= 1;// 用异或进行取反操作模拟开关灯操作，不妨思考一下它的原理
        }
    }
}
for(int i = 1; i ⩽ n; i++){
    if(flag[i] % 2)
        printf("%d ", i);
}
```

这样的做法需循环约 $\frac{n^2}{2}$ 次，效率较低，但~~温柔的~~助教gg降低了数据范围至 $n \leq 2000$，所以即使这样也可以通过。

如果你多测试几次就会发现一个规律，程序输出的正好是完全平方数的排列，事实上确实如此，接下来给出一些简单的证明：

如果对开关进行奇数次操作，最终灯保持亮的状态；如果进行偶数次操作，最终灯保持灭的状态。因此，题目所求的实际上是 1 ~ n 中因数个数为奇数的数。然而因数都是**成对出现**的：若一个数 a 的**因数个数**是奇数，说明它的因数中存在重复因数 i，即 $i * i = a$，显然 a 应该是完全平方数。

参考答案：

```
#include <stdio.h>

int n;
int main(void){
    scanf("%d", &n);
    for (int i = 1; i * i ⩽ n; i++) {
        printf("%d ", i * i); // 找规律，并不要求严格的数学证明
    }
    return 0;
}
```

Statistics

> 本题知识点：
> 整数与字符输入，格式化输出，字符数组的初始化与访问，整数数组的初始化与访问，双重循环

本题主要有将输出保存到二维数组最后一起输出和逐行输出两种解题方式，但其实两种方式代码的简洁性和思路的易懂性差异不大……

数据处理过程：题目要求对字符串中出现的各英文字母进行计数，结合上次作业中的**桶计数法**，容易想到用整型数组结合字符的 ascii 码进行计数。

数据输出过程/输出数组处理过程：程序的输出遵循从上到下，从左到右的原则，可以将样例的输出看成一张表来理解：

出现次数 ≥ 3	=			
出现次数 ≥ 2	=		=	
出现次数 ≥ 1	=	=	=	
字母	a	A	e	...

其中如果满足左侧的条件就在对应的字母上写 `=`，否则写 `␣`（空格）

这样一来思路就清晰了，只需要找到出现次数的最大值，然后从它开始向 1 遍历，每一个出现次数的值对应一行，然后再对大小写字母做一些特判即可。

同时，**强烈建议**大家不要写诸如 65 、97、122、32 等数字（即 'A'、'a'、'z'、'a' - 'A' 的 ascii 码），可读性会降低，容易记错写错，每次还要去查手册，完全没必要，直接写上**单引号包围的字符**即可。（D 题 decimal 同理）

参考答案1（朴素版）：

```c
#include <stdbool.h>
#include <stdio.h>
#define MAX(a, b) (a) > (b) ? (a) : (b)  // 宏定义，比较两数大小

char s[1005];  // 输入字符串
int n, max, len;  // 字符个数、最多字符数、分割线长度
int a[52];  // 桶，此处用偶数表示小写字母，奇数表示大写字母，具体实现见下
bool is_not_empty[52];  // 表示该字母的桶是否非空，用于计算分割线长度与是否输出字母
int main(void) {
    scanf("%d%s", &n, s);
    for (int i = 0; i < n; i++) {
        if (s[i] ⩾ 'a' && s[i] ⩽ 'z') {
            a[(s[i] - 'a') * 2]++;  // 桶计数小写字母
        }
        else if (s[i] ⩾ 'A' && s[i] ⩽ 'Z') {
            a[(s[i] - 'A') * 2 + 1]++;  // 桶计数大写字母
        }
    }
    for (int i = 0; i < 52; i++) {  // 遍历桶，步长为 1
        max = MAX(max, a[i]);  // 循环比较获得最大长度，这一步可与上一循环合并
        if (a[i] > 0)
            is_not_empty[i] = true;  // 判断该字母是否非空，这一步可与上一循环合并
    }
    for (int i = max; i > 0; i--) {  // 从最大行开始往下，逐行输出
        for (int j = 0; j < 52; j += 2) {  // 遍历桶，步长为 2，大小写同一轮处理
            if (!is_not_empty[j] && !is_not_empty[j + 1]) {
                continue;  // 如果小写字母与大写字母均为空，跳过剩下部分进入下一次循环
```

```
            }  // 逻辑上下一行之前含有 else，但由于 continue，故可省略
            for (int k = 0; k < 2; k++) {
                // k 为 0 时即小写字母，k 为 1 时即大写字母
                if (a[j + k] < i && is_not_empty[j + k]) {
                    // 如果该字母出现次数小于 i 且非空
                    printf(" ");
                } else if (a[j + k] >= i) {
                    // 如果该字母出现次数大于等于 i
                    printf("=");
                }
            }
            printf(" ");  // 仅在大小写字母并非全空的情况下才会输出空格
        }
        printf("\n");
    }
    for (int i = 0; i < 52; i += 2) {
        // 遍历非空判断数组，步长为 2，大小写同一轮处理
        if (is_not_empty[i] && is_not_empty[i + 1]) {
            len += 3;  // 如果大小写均非空 分割线长度 + 3
        } else if (is_not_empty[i] || is_not_empty[i + 1]) {
            len += 2;  // 如果大小写有一空一非空 分割线长度 + 2
        } else {
            continue;  // 这个 else 不写也行
        }
    }
    while (--len) {  // 思考一下，这样写为何是循环 len - 1 次而不是 len 次
        printf("-");  // 循环输出 len - 1 个分割线减号
    }
    printf("\n");
    for (int i = 0; i < 52; i += 2) {  // 遍历非空判断数组，步长为 2，大小写同一轮处理
        if (!is_not_empty[i] && !is_not_empty[i + 1]) {
            continue;  // 如果小写字母与大写字母均为空，跳过剩下部分进入下一次循环
        }
        if (is_not_empty[i]) {  // 如果小写字母非空
            printf("%c", i / 2 + 'a');
        }
        if (is_not_empty[i + 1]) {  // 如果大写字母非空
            printf("%c", i / 2 + 'A');
        }
        printf(" ");
    }
    return 0;
}
```

参考答案2（奇技淫巧版，23行）：

```
#include <stdio.h>
#define MAX(a, b)  (a) > (++b) ? (a) : (b)  // 特化后的比较大小，将桶计数的计数部分包含其中

int n, max, len, a[78];
// len 将从 0 逐渐加到列数，操作见下
// a[26 * 3]是为了同时存进大小写字母与空格，统一看待
char s[1005], output[100][1005];
// 将所要输出的所有东西都放入 output 二维字符串中
// 前为列，与 len 有关；后为行，与 max 有关
// 100为列数上限，1005为行数上限（都留有余地）
```

```c
int main(void) {
    scanf("%d%s", &n, s);
    for (int i = 0; i < n; i++)
        max = MAX(max, a[(s[i] - (s[i] < 'a' ? 'A' : 'a')) * 3 + (s[i] < 'a')]);
    // 这一步很有技巧性，由内向外解释：
    // s[i] - (s[i] < 'a' ? 'A' : 'a') 即判断大小写后再减去 'A' 或 'a' 的 ascii 码；
    // (s[i] - (s[i] < 'a' ? 'A' : 'a')) * 3 是为了填进 78 长度的数组；
    // 再 + (s[i] < 'a')，运用了条件为真时返回 1，此时，小写字母的下标位置为 模 3 余 0，
    // 大写字母的下标位置为 模 3 余 1，空格的下标位置为 模 3 余 2；
    // 则 a[一长串]就是需要的桶，用特化的 MAX 宏定义来计数，
    // 先 ++a[一长串] 再比较大小，返回桶计数后（加一后）与原 max 比较大小的值；
    // 这样就同时完成了桶计数与比较获取最大值。
    for (int i = 0; i < 78; i += 3) {  // 遍历桶，步长为 3
        if (!a[i] && !a[i + 1]) // 如果大小写字母都为空，跳过
            continue;
        for (int j = 0; j < 3; j++) // 遍历小写字母、大写字母、空格
            if (a[i + j] || j == 2) {
                // 如果大写或小写字母非空，或者是空格那一位，填充 output
                // k 从 max + 1 向下至 2 遍历各行，用等号或空格填充对应位置
                // 如果该字母计数超过 k - 1 便填充为等号，计数不足或是空格位则均填充为空格
                for (int k = max + 1; k > 1; k--)
                    output[len][k] = a[i + j] >= k - 1 ? '=' : ' ';
                // k 为 1 时为分割线一行
                output[len][1] = '-';
                // k 为 0 时为最下面一行，输出大小写字母
                output[len++][0] = j == 2 ? ' ' : i / 3 + (j ? 'A' : 'a');
                // len 在最后加一，进入下一空列，最外层循环结束时 len 即为总列数
            }
    }
    // 此时，我们所需要的输出便全部填充完成，直接按行按列输出即可
    // 注意列数上限为 len - 1，即去掉最后一列多余的空格/分割线
    // 这里用了 for 的 tricky 写法，其实不该这样写……
    for (int i = max + 1; i >= 0; i--, printf("\n"))
        for (int j = 0; j < len - 1; printf("%c", output[j++][i]));
    return 0;
}
```

Josephus

> 本题知识点：
>
> 整数输入，整数输出，整数（或者布尔，称标记数组）数组初始化与访问，双重循环

初学者最容易想到的思路是直接使用一个标记数组模拟题目描述的整个过程，代码比较简单就不再做过多的分析，但是问题在于在循环后期标记数组中大多数都已经被标为 0（即人已经死掉），需要跳过的节点过多，导致程序执行效率过低。不过似乎也能通过评测，只是评测用时会因为太长而被标红。

参考答案1（直接模拟题目过程 朴素版）：

```c
#include <stdio.h>

int n, k, pos, alive[501];
int main(void) {
    scanf("%d%d", &n, &k);
    for (int i = 1; i <= n; i++)
        alive[i] = 1;
```

```
    for (int i = 1; i ⩽ n - 1; i++) {
        for (int j = 1; j ⩽ k; j++) {
            pos++;
            if (pos > n) pos = 1;
            while (!alive[pos]) {
                pos++;
                if (pos > n) pos = 1;
            }
        }
        alive[pos] = 0;
    }
    for (int i = 1; i ⩽ n; i++) {
        if (alive[i]) {
            printf("%d\n", i);
            break;
        }
    }
    return 0;
}
```

参考答案2（直接模拟题目过程 01倒置精简版）：

```
#include <stdio.h>

int n, k, alive[501];
int main(void) {
    scanf("%d%d", &n, &k);
    for (int p = 1, i = n; i > 1; i--) {
        for (int j = 1; j ⩽ k; p = p % n + 1) {
            if (alive[p])
                continue;
            if (j == k)
                alive[p] = 1;
            j++;
        }
    }
    for (int i = 1; i ⩽ n; i++) {
        if (!alive[i]) {
            printf("%d\n", i);
            break;
        }
    }
    return 0;
}
```

提高效率的办法主要有两种，其一是利用另外一个数据结构（链表）的性质，在课程的后期会讲到它，这里就不作展开；

其二是通过一些~~简单的~~**数学分析**将问题简化。

假设所有人的编号为 $0, 1, 2, \ldots, n-1$，设$f(n, m)$表示n个人玩$k = m$的约瑟夫游戏剩下的幸存者，有递推公式：

$$f(n, m) = (f(n-1, m) + m) \ mod \ n$$

推导过程可参考：乔瑟夫问题推导过程

但由于这并不是算法课和数学课，所以如果看不懂也没有关系，你甚至可以直接跳过这一部分

参考答案3（数学分析简化）：

```c
#include <stdio.h>

int n, k, ans = 1;
int main(void) {
    scanf("%d%d", &n, &k);
    for (int i = 2; i <= n; i++)
        ans = (ans + k - 1) % i + 1; // 递推，并不要求掌握
    printf("%d\n", ans);
    return 0;
}
```

Decimal

> 本题知识点：
> 字符数组输入，整数输出，字符数组初始化与访问，字符转整数，单重循环

简单的进制转换题目，基本思路可分为边循环边用计数器乘 N、计算前使用数组初始化 N^i、和在计算过程中直接使用 $pow(N,i)$ 三种思路。

总体上需要注意的点只有 i 与读入数组下标的对应关系、前导0 和 全0 的处理方式和 G~Z 的大写字母导致输入不合法。

参考答案：

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

int len, n, ans;
char s[32];
int main(void) {
    scanf("%d%d%s", &len, &n, s);
    for (int i = 0; i < len; i++)
        if (s[i] <= '9' && s[i] - '0' >= n ||
            s[i] >= 'A' && s[i] - 'A' + 10 >= n) {
            // 在字符串状态下先判断是否合法，不要写诸如 48、65、55 等数字，可读性差！
            printf("Error");
            return 0; // 此处有很多同学不知道如何退出程序，可以这样写
        }
    for (int i = 0; i < len; i++)
        if (s[i] <= '9')
            ans += (s[i] - '0') * pow(n, len - i - 1);
            // pow 会自动将类型从 double 转化成 int
            // 也可以不用 pow，循环一次乘一次即可
        else
            ans += (s[i] - 'A' + 10) * pow(n, len - i - 1);
    printf("%d", ans);
    return 0;
}
```

Palindrome

> 本题知识点：
> 整数与字符串输入，字符串输出，字符数组初始化，访问与赋值，双重循环

本题的解决思路也主要分为两种，分别是记录双边以及需要补上的数据值、记录补好的数组两个方式。

首先需要处理的是将单边问号，即对称位置上不全为 `?`，补充好对应的值；

其次需要处理的是如果是双边问号，即对称位置上全为 `?`，那么就用一个计数器记录当前情况的编号，编号自增，这个编号就是需要补充的数值；

最后需要处理的是如何输出，如果采用第一种方式，就需要用一个标记数组记录在字符串下标为多少的时候出现了这个情况，在这种情况下需要补充的编号是多少，或者在输出的过程中直接处理；如果采用第二种方式，可以考虑用另一个数组存储补充好的字符串的前半部分，后半部分倒序输出，并特判奇偶的情况（较为复杂）。

总之，边填问号边输出可以大大化简本题的代码量与代码难度，如果想要整合进一个字符串再统一输出会更加复杂。

由于上述两种方式的差别并不是太大，这里只给出第一种方法的参考答案：

```c
#include <stdio.h>

int n;
char s[100005];
int main(void) {
    scanf("%d%s", &n, s);
    for (int i = 0; i < n; i++) {
        if (s[i] == '?')
            s[i] = s[n - 1 - i]; // 仅处理单边问号，双边问号留给下一步
    }
    int cnt = 0;  // 计数器，从 "00" 开始
    for (int i = 0; i < n; i++) {
        if (s[i] == '?' && i < n / 2) {
            // 处理前半段的双边问号并直接输出，注意前半判定的条件
            printf("%02d", cnt); // 注意输出两位数的方法
            cnt++; // 计数器加一
        } else if (s[i] == '?' && i >= n / 2) {
            // 处理前半段的双边问号并直接输出，注意后半判定的条件
            cnt--; // 因为前半走完时，计数器多加了一次，故后半先减一再输出
            printf("%02d", cnt / 10 + (cnt % 10) * 10); // 对调十位个位
        } else {
            printf("%c", s[i]); // 若不是问号，直接输出
        }
    }
    return 0;
}
```

Quick-sort

> 本题知识点：
> 整数输入，while循环，for循环，双重循环，条件分支语句

一道对所有人都极不友好的题目是如何诞生的?

> （前一周的出题会议上）
>
> 🐜：去年我们出的是选择排序，今年稍微改一下，改成冒泡排序吧
>
> 众助教：（纷纷赞同）
>
> zzk：（会议结束后2h）码字ing……冒泡排序已经出好了，欢迎大家验题:）
>
> （two days later……）
>
> 🐜：技科班有的老师在课上已经讲过冒泡排序了，我们可能需要换一个题 （附件：bubble-sort.png）

zzk：我出的冒泡排序顺序和老师讲的顺序不太一样，实在不行就换个题吧

🐜：……感觉如果只是改变一下顺序学生稍微改一下代码就可以了，要不我们出成双向冒泡排序或者介绍一下快排的 partition 的部分

zzk：我倾向于改快排（one day later……）快速排序已经出好了，大家快来验验题 :）

czh：我觉得你的描述不太清楚，我给你改一下你看看……（a few moments later）我觉得现在很清楚了！但是感觉还是要多来点助教验下题……

这个垃圾题，助教自己出完也不满意，因为做法卡得太死，出于介绍算法思想考虑，又因为快速排序思想比较绕，为了帮助初学者直接能够实现而不用多花很多精力去思考它的原理还不好加 spj，于是出成了一道模拟题，但大家似乎并没有意识到这是一道模拟题，而是选择根据自己的思想来写~~遍地开花~~，也有可能是因为看出来了是道模拟题，但自己的实现并没有表达出题目的意思。

接下来我们一句一句地分析题目，带大家完成这个程序的实现：

排序过程中，首先使用两个变量 l 和 r ，l 按 $0,1,2,\ldots,n-1$ 的顺序从左向右遍历数组下标，初始为 0 ，而 r 按 $n-1,n-2,n-3,\ldots,0$ 的顺序从右向左遍历下标，初始为 $n-1$。排序过程如下：

1. l 从当前位置向右遍历，直到 $l==r$ 或遇到一个 $\geq pivot$ 的数。
2. r 从当前位置向左遍历，直到 $l==r$ 或遇到一个 $<pivot$ 的数。
3. 如果 l 和 r 不相等 ，则交换 a_l 和 a_r 的值。

```
int a[1000005];
int l = 0, r = n - 1, pivot = a[k - 1]; //l 初始为 0，r 初始为 n-1，pivot 取值
while (l ≠ r && a[l] < pivot) l++; // l 从当前位置向右遍历，直到 l==r 或遇到一个 ≥pivot 的数
while (l ≠ r && a[r] ⩾ pivot) r--; // r 从当前位置向左遍历，直到 l==r 或遇到一个 <pivot 的数
if (l ≠ r){ // 如果 l 和 r 不相等，则交换 a[l] 和 a[r] 的值
    int tmp = a[l];
    a[l] = a[r];
    a[r] = tmp;
}
```

4. 如果 l 和 r 不相等，继续从步骤 1 开始往下执行，直到 l 与 r 相等为止。

```
while (l ≠ r){ // 如果 l 和 r 不相等，就重复执行上述 1~3 步
    ... // 上面从第 3 行开始的内容
}
```

在 l 和 r 相遇以后，当前的数组 a 仍可能不满足本题中的要求。为满足要求，还需对 a 中的两个元素进行至多一次交换，不妨来看看样例2，当 l 已经找到一个大于等于 $pivot$ 的数，但 r 并未找到一个小于 $pivot$ 的数就已经与 l 相等了的时候，在 $pivot$ 的左边仍然有一个大于等于它的数，这时就需要将 l 这个位置上的数与 $pivot$ 对应位置的数相交换。

你可能还想问：如何才能找到 $pivot$ 现在在哪个位置呢？这里给出最简单的做法：

既然 $pivot$ 的值在数组中仅出现一次，在算法开始之前记录下它的值，然后在循环结束后再遍历一遍数组，看看和它相等的数在哪个位置就可以了

```
if(a[l] > pivot){
    //如果循环结束当前的数组仍不满足要求，即 l 找到了一个大于等于 pivot 的数，但 r 直接和 l 相遇了
```

```
    for (int i = 0; i < n; i++){
        if (a[i] == pivot){
            // 找到 pivot 现在在数组中的位置，因为在前面的交换过程中，
            // l 可能找到了 pivot，r可能找到了一个小于 pivot 的数，然后它们进行交换，
            // 则 pivot 就不再位于数组的 k-1 位置上
            int tmp = a[l];
            a[l] = pivot;
            a[i] = tmp;
            // 交换 a[l] 和 pivot 对应位置上的值
            break;// 因为 pivot 只出现一次，所以找到之后循环就可以结束了，不加这一句也没关系
        }
    }
}
```

以上，这道题便做完了，从结果来看其实不难，但从做题的角度来说确实是卡得太死了，完全要学生去猜。（就算补全题面后也是看得不明不白，太垃圾了！）

做完了题目，你可能仍有疑惑：快速排序为什么是正确的？在这里我们给出一个形象的理解方式（不是证明）：在一趟排序之后，我们保证了存在某个下标 p ，使得 $a[0:p]$ 和 $a[p+1:n-1]$ 两个区间中任何两个数有序，如果我们进行很多趟，每趟都选取合适的区间 ，直到数组中的每一个 $a[i:i]$ 和 $a[i+1:i+1]$ 都有序，那么排序就完成了。

多位助教一致认为，无论如何，有关快速排序的性质（如正确性、时间、稳定性等问题）都不是一个在面向 0 基础的第四节课的作业中应该思考的问题，请同学们不必纠结，随着学习深入自会对这些问题有更深刻的理解。助教在答疑时也只会点到即止（事实上，有些太过于高深的问题助教也搞不清楚 QAQ）。

参考答案：

```
#include <stdio.h>

int n, k, a[1000005];
int main(void) {
    scanf("%d%d", &n, &k);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    int l = 0, r = n - 1, pivot = a[k - 1];
    while (l ≠ r) {
        while (l ≠ r && a[l] < pivot) l++;
        while (l ≠ r && a[r] ⩾ pivot) r--;
        if (l ≠ r) {
            int tmp = a[l];
            a[l] = a[r];
            a[r] = tmp;
        }
    }
    if (a[l] > pivot) {
        for (int i = 0; i < n; i++) {
            if (a[i] == pivot) {
                int tmp = a[l];
                a[l] = pivot;
                a[i] = tmp;
            }
        }
    }
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

此外，关于交换数组中的两个数，有多种方法。

上述为最直白的写法，记为法一，下面会给出几种更为简单方便的写法（原理需自行探究）：

法二 宏定义：

```
#define SWAP(a, b) \
    do {           \
        int t = a; \
        a = b;     \
        b = t;     \
    } while (0)
// 宏需写在代码头文件下面，此为简单版，交换两个 int 型数字
int main(void){
    int x, y;
    // some codes...
    SWAP(x, y); // 使用方法
    // some codes...
}
```

法三 指针 + 函数：

```
void swap(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}
// 原理在学到函数（function）传参与指针（pointer）后便会知晓
int main(void){
    int x, y;
    // some codes...
    swap(&x, &y); //使用方法
    // some codes...
}
```

法四 异或赛高：

```
int main(void){
    int x, y;
    // some codes...
    x ^= y ^= x ^= y; // 异或真神奇！不过这究竟是为什么呢？
    // some codes...
}
```