# CPL 第二次编程练习 2-if-for-array 题解

教学周历中本周课程的知识点为：

> if 语句；for 循环语句（初步）；一维数组

教材章节：

> 5.1–5.3；6.3；8.1

备注：

> 软件学院 2 个学时内讲不了 switch/case（5.3），可以安排学生自学。

作业中不会涉及知识点范围之外的内容，如果你使用了超纲的知识，不妨来看一下标答与解析。

如果你是因为自学了后面的内容而感觉作业超纲，那么请你好好阅读本周的题解。正如每道高考题都有超纲做法，但也总有课标内的做法。不怕慢鸟先飞，最怕一知半解。

## Single

> 本题知识点：
> 读入一串已知数量的整数，for 循环，整数输出，整数数组初始化，访问与赋值/整数基本运算（软件学院班级可能会使用计算系统基础上学习的异或运算）。

对于初学者，可能~~容易~~想到二重循环的做法。

```
for (int i = 0; i < n; i++) {
  bool once = true;
  for (int j = 0; j < n; j++) {
    if (i != j && a[i] == a[j]) {
      once = false;
    }
  }
  if (once) printf(...)
}
```

这样的做法需要循环约 $n^2$ 次，效率太低，无法通过，事实上，在 `a[]` 中的值不大的情况下，可以使用“桶计数法”。

具体来说，可设 `int bucket[MAX_A]` 来表示（其中 `MAX_A` 表示 `a[]` 中出现的最大的值 $+1$），`bucket[i]` 表示 `i` 这个数在 `a[]` 中出现了 `bucket[i]` 次。

在循环读入 `a[i]` 的过程中，每读入一个 `a[i]`，就执行对应的 `bucket[a[i]]++`，在读入完成后，就得到了相应的 `bucket[]` 的值。最后只需要遍历观察哪个 `i` 满足 `bucket[i] == 1`，输出那个 `i` 即可。

参考答案：

```
#include <stdio.h>
#define BUCKET_SIZE 1000000

int n, a, bucket[BUCKET_SIZE];

int main(void){
    scanf("%d", &n);
    for (int i = 0; i < 2 * n - 1; i++) {
        scanf("%d", &a);
        bucket[a]++;
    }
```

```
    for (int i = 0; i < BUCKET_SIZE; i++) {
        if (bucket[i] == 1) {
            printf("%d", i);
            break;
        }
    }
    return 0;
}
```

本题还有一种巧妙的做法，需要使用到异或运算（在 C 语言中写作 `^`，具体原理会在《计算系统基础》与《计算机组织结构》两门课中学到）。

```
for (int i = 0; i < n; i++) {
  scanf("%d", &x);
  ans = ans ^ x;
}
```

该做法技巧性太强，不易扩展，不过多展开。

## Factorial

> 本题知识点：
> 整数输入，整数输出，循环，整数基本运算，取余运算的分配律（调整 n 的大小，使得 sum of n! 超出 long long 的范围。）

使用循环来计算阶乘。

```
int fac = 1; // 0! = 1
for (int i = 1; i <= n; i++) {
  fac = fac * i;
}
```

需要注意的是 $n!$ 可能非常大，远远大于 `int` 甚至 `long long` 的范围。例如，$25! = 15511210043330985984000000$，而 `long long` 的上界则通常为 $2^{63} - 1 = 9223372036854775807$。

因此，在循环完成之后再对结果进行取模是不正确的，我们可以在循环运行的过程中不断取模，取模的数学性质（见提示）保证了这样操作的正确性。

```
const int Mod = 10007;
int fac = 1;
for (int i = 1; i <= n; i++) {
  fac = fac * i % Mod;
}
```

需要注意的是，有很多同学将代码写成了以下形式。

```
const int Mod = 10007;
int fac = 1;
for (int i = 1; i <= n; i++) {
  int tmp = i % Mod;
  fac = fac * tmp;
}
```

这样并没有解决 `fac` 变量的溢出问题，事实上，在 $1 \leq i \leq n \leq 25$ 的情况下，$i \mod 10007 = i$。

参考答案1：

```c
#include <stdio.h>

const int Mod = 10007;
int n, fac = 1, sum = 0;

int main(void) {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        fac = fac * i % Mod;
        sum = (sum + fac) % Mod;
    }
    printf("%d", sum);

    return 0;
}
```

为了计算阶乘的和，需要对于每个 $i \leq n$ 计算 $i!$，双重循环或许更易理解，但也可利用阶乘的“递推性”，即 $(n!) = (n-1)! \times n$。

参考答案2：

```c
#include <stdio.h>

const int Mod = 10007;
int n, sum = 0;

int main(void) {
    scanf("%d", &n);
    for (int i = n; i >= 1; i--) {
        sum = (sum + 1) * i % Mod;
    }
    printf("%d", sum);
    return 0;
}
```

想要了解更多有关数据类型的知识？可以参考文档或《计算系统基础》的课本。

## Series

> 本题知识点：
> 数学公式，循环，浮点数计算

按照公式循环计算。本题基本没有精度问题，既可以使用 `pow`，也可以使用循环来计算 $x^{2i+1}$。

可能需要说明的是求和号 $\sum$ 的意义，

$\sum_{i=l}^{r} a_i = a_l + a_{l+1} + \cdots + a_r$

参考答案：

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

int n;
double x;
int main(void) {
    scanf("%lf%d", &x, &n);
```

```
    double ans = 0, cur = x;
    for (int sgn = 1, i = 0; i <= n; i++, sgn *= -1) {
        ans += 1.0 * sgn / (i + i + 1) * pow(x, i + i + 1);
    }
    printf("%.10f", 4 * ans);
    return 0;
}
```

## Collatz

> 本题知识点：
> 整数输入、条件语句、for 循环

在“循环”这一节中经常出现的练习题，需要弄清楚循环停止的条件。

99% 的 `for` 循环往往是以下形式，

```
for (int i = 1; i <= n; i++)
```

但 `for` 循环可以使用得更“花哨”一点，或许可以这么理解 `for` 循环：

```
for (do sth. before loop; condition when loop ends; do sth. iteratively)
```

比如，我们可以这样从两端遍历数组：

```
for (int i = 0, j = n - 1; i < j; i++, j--)
```

再比如，Sa 酱在第二次作业-落单的数一题中的极致压行程序在读入处理数据时（看不懂没关系，一般不会这样写）：

```
for (scanf("%d", &x); scanf("%d", &x) != EOF; ans ^= x); // 先读入 x 是为了处理垃圾值
```

因此在这里我们可以这样写：

```
for (steps = 0; n != 1; steps++)
```

**请注意：这只是理解 `for` 循环的一种方式，我们提倡简洁优雅易读的代码风格，比如第二个读入处理数据的例子，就是一个典型的不易读的例子，不要学。**

参考答案：

```
#include <stdio.h>

int n, max, steps;

int main(void) {
    scanf("%d", &n);
    max = n;
    for (steps = 0; n != 1; steps++) {
        if (n % 2 == 1)
            n = n * 3 + 1;
        else
            n = n / 2;
        if (n > max)
            max = n;
    }
```

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

写题解的助教认为，这里还是 `while` 循环更加容易理解，但这周课上没有讲过 `while` 循环， `for` 循环和它们的表达能力是等价的。

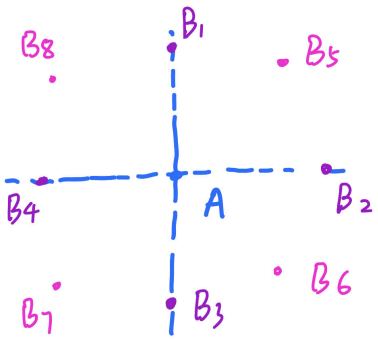
## Obstacle

> 本题知识点：
> 整数输入，条件语句，for 循环

### 一道对初学者极不友好的题目是如何诞生的?

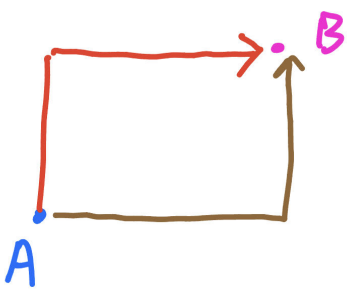
> （前一周的出题会议上）
>
> 🐜：去年的老题都改的差不多了，大家有没有新题呢?
>
> czh：老师老师，我有一道题。如果给定二维平面上的起点和终点，然后再给定一个不能走的障碍点，求最短路径长度。
>
> 🐜：这个题可以，但是我们下周的主题是 `if-for-array` ，有没有办法用上 `for` 循环呢?
>
> 众助教：……
>
> 🐜：如果输出路径，能做吗?
>
> czh：…… 能做。
>
> （第二天，写完了标程的我开始意识到事情的不对劲……）
>
> czh：（试图挽回）这周作业有点难，老师要不我们加个简单题吧……
>
> 🐜：不通过加简单题的方式降低作业难度。要不把最短路径的输出路径拿了?
>
> czh：(╥╯^╰╥) 数据我可是造了三个小时……

首先考虑 A 和 B 的位置关系，具体来说有 8 种情况：




简单地分一分，可以分为 A、B 在一个矩形的对角线上的情况和 A、B 在一条直线上的情况。

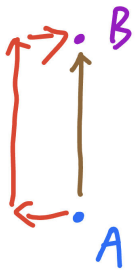
先考虑 A、B 在一个矩形的对角线上的情况，



容易发现用红色笔和褐色笔画出的两条最短路径，事实上，还有一些其他的最短路径，但本题中只考虑其中最特殊的两条。因为这两条最短路径是不相交的，如果向这个图上放一个障碍物的话，最多只能挡住一条。只需使用 `if` 语句考察 C 点是否在褐色的路径上，如是，则走红色路径，否则直接走褐色路径。

这种情况下最短路径为 $|x_a - x_b| + |y_a - y_b|$。

然后考虑 A、B 在一直线上的情况。




这种情况下最短路径为褐色路径，答案也为 $|x_a - x_b| + |y_a - y_b|$。

但若 C 点出现在褐色路径上，则需要向旁边绕一步，如红色路径所示，答案为 $|x_a - x_b| + |y_a - y_b| + 2$。

更详细的解释？请参考 10 月 11 日晚的视频讲解。

参考答案：

```c
#include <stdio.h>
#include <stdlib.h>  // abs(x)

int xa, ya, xb, yb, xc, yc;

int main(void) {
    scanf("%d%d%d%d%d%d", &xa, &ya, &xb, &yb, &xc, &yc);
    int ans = abs(xa - xb) + abs(ya - yb);
    if (xa != xb && ya != yb) {
        printf("%d\n", ans);
        if (xc != xb && yc != ya) {
            for (int x = xa; x != xb; x += (xb > xa) ? 1 : -1) {
                if (xb > xa)
                    printf("R");
                else
                    printf("L");
            }
            for (int y = ya; y != yb; y += (yb > ya) ? 1 : -1) {
                if (yb > ya)
                    printf("U");
                else
                    printf("D");
            }
        } else {
            for (int y = ya; y != yb; y += (yb > ya) ? 1 : -1) {
                if (yb > ya)
                    printf("U");
                else
                    printf("D");
            }
            for (int x = xa; x != xb; x += (xb > xa) ? 1 : -1) {
                if (xb > xa)
                    printf("R");
                else
                    printf("L");
            }
```

```
        }
    } else if (xa == xb) {
        int l = ya, r = yb, flag = 0;
        if (l > r)
            l = yb, r = ya;
        if (xc == xa && yc > l && yc < r) {
            flag = 1;
            ans += 2;
        }
        printf("%d\n", ans);
        if (flag)
            printf("R");
        for (int y = ya; y != yb; y += (yb > ya) ? 1 : -1) {
            if (yb > ya)
                printf("U");
            else
                printf("D");
        }
        if (flag)
            printf("L");
    } else {
        // ya == yb
        int l = xa, r = xb, flag = 0;
        if (l > r)
            l = xb, r = xa;
        if (yc == ya && xc > l && xc < r) {
            flag = 1;
            ans += 2;
        }
        printf("%d\n", ans);
        if (flag)
            printf("U");
        for (int x = xa; x != xb; x += (xb > xa) ? 1 : -1) {
            if (xb > xa)
                printf("R");
            else
                printf("L");
        }
        if (flag)
            printf("D");
    }
    return 0;
}
```

## Inverse2

> 本题知识点：
> oj-1-types-io 的扩展，输入 n（长度）与 k（反转位置）：定长字符串输入（包含空白符）、for 循环、数组

本题旨在帮助大家理解 C 语言中~~复杂~~的输入行为。

### 如何读入字符串 / 字符?

我们在此进行简单的归纳。假设已经定义了 `str[LEN]`。

- `scanf("%s", str)`：读入一个字符串（与 `%d` 不同，`%s` 所对应的 `str` 前通常不加 `&`），但读到空格就会停止。因此本题不能简单地使用它，在输入的字符串没有空格的时候推荐大家使用。
- `for (...) str[i] = getchar()`：读入一个字符，理论上推荐大家使用，但初学者可能难以理解字符系统的显示行为，因此可能会产生与想象不一致的后果。

- `for (...) scanf("%c", &str[i])`：读入一个字符，基本于 `getchar()` 等同，但在实际中似乎使用不多。
- `gets(str)`（`fgets` 等）：读入一行，`gets` 有历史渊源，不推荐大家使用。

想要了解具体的行为？当然是阅读scanf 的手册 和 getchar 的手册 。

## 机器眼中的输入

人眼中的输入可能是这样的。

```
5
abcde
3
```

但机器（linux）眼中的输入就是一串连续的字符（数据）。

```
5'\n'abcde'\n'3
```

其中 `\n` 是一个字符，代表换行。机器在显示时解析这一串字符，将人眼认识的换行体现出来。

如果在读入 $n$ 之后直接采用循环 + `getchar() / scanf("%c")` 的方式来读入数组的话，就会首先读入换行符，因此，需要使用一个 `getchar()` 或 `scanf("%c")` 来首先读掉这个换行符，之后才能采用循环读入一个字符的方法读入。

## 为什么 scanf("\n") 不行?

`scanf("\n")` 的行为并不只是读一个换行符，事实上，`scanf` 将 `'\n'`，`' '` 等字符都视为空白符（whitespace），格式串里的一个空白符会匹配输入中连续的若干个的空白符，如果下一行中的字符串以空白符开头，就发生了问题。

想要获取更详细的解释？请参考 `scanf` 的手册 。

参考答案：

```c
#include <stdio.h>

char str[10000000];
int n, k;

int main(void) {
    scanf("%d", &n);
    getchar();
    for (int i = 0; i < n; i++) {
        scanf("%c", &str[i]);
    }
    scanf("%d", &k);

    for (int i = 0; i < k; i++) {
        printf("%c", str[k - i - 1]);
    }
    for (int i = k; i < n; i++) {
        printf("%c", str[n - 1 + k - i]);
    }
    return 0;
}
```