让程序有规律地重复

从计数、累加到嵌套循环与数位处理

HGNU-ACM 实验室算法组

基础篇 + 18 道例题逐题推导，49 个练习位置，43 道不同练习题

本章目录

基础一：从重复动作到 for 循环

基础二：while 与 do-while

基础三：break、continue 与嵌套循环

基础四：数位、质数与输出模板

取余、因子与状态更新

嵌套循环与枚举

本章全部练习与学校 OJ

为什么需要循环？

- 如果让程序算 $1+2+\cdots+100$，手写 100 条加法语句既笨拙又容易错。
- 循环把“重复做同一件事”交给计算机：写一次，执行很多次。
- 写循环前先回答三个问题：从哪里开始、做到什么时候停、每轮怎样变化。



本章主线

计数 → 累加/累乘 → 条件未知的重复 → 两层循环。

for 的语法：三个部分



- 三个表达式用两个分号隔开；初始化只在进入循环前做一次。
- 每轮先判断条件，为真才进入循环体。
- 循环体执行完，做更新，再回到条件处。

初学建议

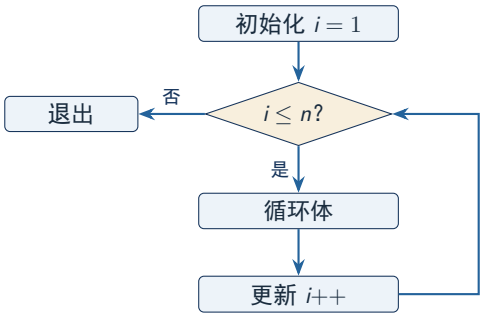
三个部分都写全，不要省略；省略后很容易看不清循环怎样结束。

for 的执行流程

1. 执行初始化（只一次）。
2. 判断条件：假就退出。
3. 条件为真，执行循环体。
4. 执行更新，回到第 2 步。

记住

条件一开始就为假时，循环体一次也不执行。



手算跟踪：$n = 4$ 的求和

```
int sum = 0;
for (int i = 1; i <= 4; i++) {
    sum += i;
}
```

轮次	判断前的 i	条件 $i \le 4$	执行后 sum
1	1	真	1
2	2	真	3
3	3	真	6
4	4	真	10
5	5	假	退出

结论

循环体执行 4 次；退出时 $i = 5$，$sum = 10$。

循环范围与步长

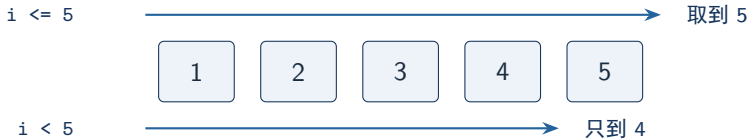
写法	i 的取值	次数
`for (int i = 1; i <= n; i++)`	$1, 2, \ldots, n$	n
`for (int i = 0; i < n; i++)`	$0, 1, \ldots, n-1$	n
`for (int i = 2; i <= n; i += 2)`	$2, 4, \ldots$	$\lfloor n/2 \rfloor$
`for (int i = n; i >= 1; i--)`	$n, n-1, \ldots, 1$	n

- 步长可以为负：`i--` 或 `i -= 2`。
- 更新必须让条件最终变假，否则会死循环。

空循环

当 $n = 0$ 时，`i <= n` 一开始就为假，循环体一次也不执行。

边界：i < n 还是 i <= n



- 从 1 数到 n：最后一项是 n，条件写 `i <= n`。
- 从 0 数到 $n-1$：最后一项是 $n-1$，条件写 `i < n`。
- 两种写法都执行 n 次，但循环变量退出时的值不同。

自检

把 $n=1$ 代进去：`i <= 1` 执行一次，`i < 1` 一次也不执行。

计数、累加与累乘

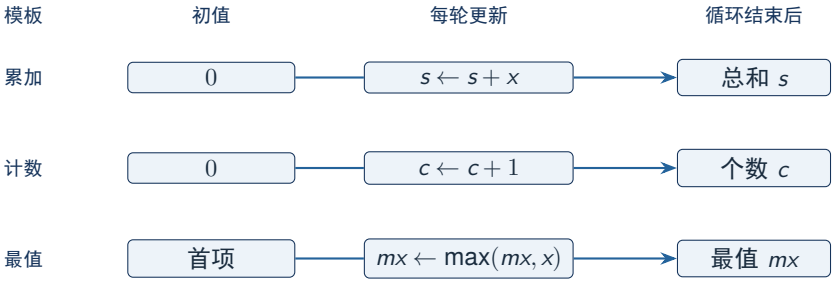
任务	变量初值	每轮更新	例子
计数	0	$c \leftarrow c+1$	数一数有几个偶数
求和	0	$s \leftarrow s+x$	年龄总和
求积	1	$p \leftarrow p \times x$	阶乘
最值	首项	$\min$ / $\max$	最高分

- 加法单位元是 0，乘法单位元是 1；初值写错，后面全错。
- 最值的初值要么取第一个数据，要么取“正无穷/负无穷”。

常见错误

把乘积初值写成 0，结果永远是 0；把最大值初值固定为 0，全负数时会错。

三个模板的共同形状



- 三个模板都是“设初值 $\rightarrow$ 循环更新 $\rightarrow$ 输出结果”，区别只在初值和更新式。
- 先确定“每轮结束时哪个量代表当前答案”，再写循环。

记住

模板不是背代码，而是记住“状态怎样随每一轮变化”。

模板一：累加器

```
int n, x, sum = 0;                  // sum 从 0 开始
cin >> n;
for (int i = 0; i < n; i++) {
    cin >> x;                       // 读入一个数
    sum += x;                       // 累加到 sum
}
cout << sum << '\n';
```

手算

输入 3 5 2 4：*sum* 依次为 $0 \to 5 \to 7 \to 11$，输出 11。

为什么初值是 0

加法单位元是 0；若写成 1，答案会凭空多 1。

模板二：计数器

```
int n, x, cnt = 0;                  // cnt 记录个数
cin >> n;
for (int i = 0; i < n; i++) {
    cin >> x;
    if (x % 2 == 0) {
        cnt++;                      // 找到一个偶数
    }
}
cout << cnt << '\n';
```

- 每满足一次条件，计数器加 1；不满足就不加。
- 输入 3 5 2 4 时，前两个数不计数，后两个数让 *cnt* 变成 2。

易错

把 cnt++ 写在 if 外面，会把所有数都数进去。

模板三：最大值与最小值

```
int n, x;
cin >> n >> x;
int mx = x, mn = x;              // 第一个数作初值
for (int i = 1; i < n; i++) {
    cin >> x;
    mx = max(mx, x);             // 更新最大值
    mn = min(mn, x);             // 更新最小值
}
cout << mx << ' ' << mn << '\n';
```

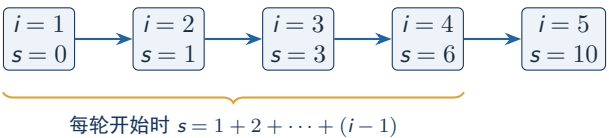
- 初值取第一个数，保证“当前最值”一定来自已经读过的数据。
- 若 n 可能为 0，改用 INT_MAX 与 INT_MIN，并特判无数据。

易错

最大值初值固定写 0，遇到全是负数时会输出错误的 0。

循环不变量（一）：定义与三问

- 循环不变量：在每一轮循环开始时都成立的断言。
- 检查三件事：初始化（第 1 轮前成立）、保持（一轮后仍成立）、终止（退出时给出答案）。



读图

方框里是“进入这一轮时”的变量值；$i=5$ 时条件为假，循环结束。

循环不变量（二）：最大值模板

- 断言：每轮开始时，mx 等于前 $i-1$ 个数中的最大值。
- 初始化：$i=1$ 时先把第一个数放进 mx。
- 保持：读入 x 后与 mx 比较，必要时更新，mx 仍是前 i 个数的最大值。
- 终止：$i=n+1$，mx 是全部 n 个数的最大值。

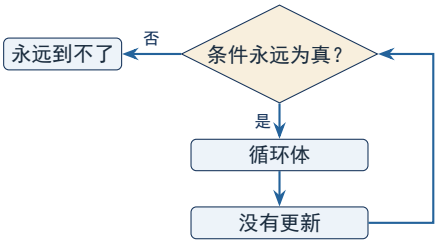
阶段	要验证的内容
初始化	$i=1$ 时 $mx=a_1$
保持	一轮后 $mx=\max(a_1,\ldots,a_i)$
终止	$i=n+1$ 时 $mx=\max(a_1,\ldots,a_n)$

作用

循环不变量能解释“为什么这样写是对的”，也能帮你定位边界错误。

死循环：条件永远为真

- 忘记更新循环变量：`i++` 漏写。
- 更新方向反了：条件要变小，变量却一直变大。
- 条件本身永远为真，循环体里又没有 `break`。

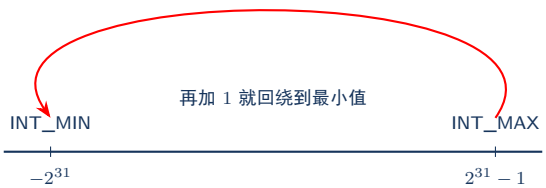


改法

让循环体每轮都朝“条件变假”的方向前进一步；必要时加 `break` 作为出口。

溢出：累加器装不下怎么办

类型	大约上限
`int`	2.1×10^9
`unsigned`	4.3×10^9
`long long`	9.2×10^{18}
`unsigned long long`	1.8×10^{19}



例子

$1+2+\cdots+10^5 = 5.0 \times 10^9$ 超过 `int` 上限，累加和要改用 `long long`。

循环里的输入处理（已知个数）

```
int n;
cin >> n;                          // 先读入数据个数
long long sum = 0;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;                      // 每轮读一个数
    sum += x;
}
cout << sum << '\n';
```

- 输入格式是“先给个数 n，再给 n 个数”时，用计数循环最自然。
- `cin >> x` 会自动跳过空格和换行，不需要自己处理分隔符。

不要多读

循环恰好执行 n 次；多读一次会吃掉下一组数据。

写循环的四步检查法



1. 初值：累加写 0，累乘写 1，最值写第一个数。
2. 条件：写完后问“最后一次执行时变量是多少”。
3. 更新：每轮都必须让条件更接近变假。
4. 边界：代入 $n = 0, 1, 2$ 手算，检查一次也不执行的情况。

口诀

初值、条件、更新、边界，四样都对，循环才对。

循环次数与复杂度直觉

n	一层循环约 n 次	两层各 n 次约 n^2 次
10	10	100
100	100	10^4
1000	1000	10^6
10^5	10^5	10^{10}

- 一般把 10^8 次简单运算看作 1 秒左右的上限。
- 两层循环都是 n 次时，总次数是 $n \times n$，不是 $2n$。
- 能由总数直接算出的量，就不要再多写一层枚举。

估算

看到 $n \le 10^5$，就要警惕 n^2 的写法；看到 $n \le 1000$，n^2 通常可行。

【例 4.1】for 循环求和 | 如何推导

1. 用变量 s 记住目前已经加出的和，初值为 0。
2. 循环变量 i 从 1 到 n，每次把 i 加到 s。
3. 循环结束时 $s = 1 + 2 + \cdots + n$。

【例 4.1】for 循环求和｜样例与边界

手算过程

$n = 10$ 时，s 依次变为 $1, 3, 6, \ldots, 55$，答案 55。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.1】for 循环求和 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    long long sum = 0;          // 用 long long 防止溢出
    for (int i = 1; i <= n; i++) {
        sum += i;               // 累加第 i 项
    }
    cout << sum << '\n';
    return 0;
}
```

【例 4.2】输出偶数 | 如何推导

1. 偶数依次是 $2, 4, 6, \ldots$，因此循环步长可以写 2。
2. 只有 $i \leq n$ 时才输出，$n = 1$ 时循环一次也不执行。
3. 用一个标志控制空格，使相邻数字之间恰好一个空格。

【例 4.2】输出偶数｜样例与边界

手算过程

$n = 10$ 时，循环变量依次为 $2, 4, 6, 8, 10$，输出 `2 4 6 8 10`。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.2】输出偶数 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    bool first = true;          // 控制空格
    for (int i = 2; i <= n; i += 2) {
        if (!first) {
            cout << ' ';
        }
        cout << i;              // 输出当前偶数
        first = false;
    }
    cout << '\n';
    return 0;
}
```

【例 4.3】输出奇偶数之和 | 如何推导

1. 准备奇数和、偶数和两个累加器，初值都为 0。
2. 枚举 1 到 n；用 $i \bmod 2$ 判断该加入哪一个和。
3. 题目要求先输出偶数和，再输出奇数和，注意顺序。

【例 4.3】输出奇偶数之和 | 样例与边界

手算过程

$n = 10$ 时，偶数和 $2 + 4 + 6 + 8 + 10 = 30$，奇数和 $1 + 3 + 5 + 7 + 9 = 25$。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.3】输出奇偶数之和 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    int odd = 0, even = 0;       // 两个累加器
    for (int i = 1; i <= n; i++) {
        if (i % 2) {
            odd += i;            // 奇数
        } else {
            even += i;           // 偶数
        }
    }
    cout << even << ' ' << odd << '\n';
    return 0;
}
```

【例 4.4】求阶乘 | 如何推导

1. 阶乘是连续的乘法：$n! = 1 \times 2 \times \cdots \times n$。
2. 乘积变量初值必须为 1，否则乘上任何数都得到 0。
3. $20!$ 很大，需要使用 `unsigned long long`。

【例 4.4】求阶乘 | 样例与边界

手算过程

$n = 4$：f 从 1 依次乘 $2, 3, 4$，得到 24。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

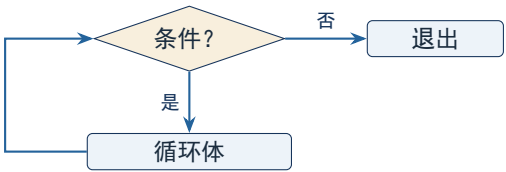
【例 4.4】求阶乘 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    unsigned long long f = 1;   // 阶乘初值为 1
    for (int i = 2; i <= n; i++) {
        f *= i;
    }
    cout << f << '\n';
    return 0;
}
```

while：先判断，再执行

```
int i = 1;
while (i <= n) {
    sum += i;
    i++;          // 靠近终止
}
```



- 条件为真才进入循环体。
- 条件一开始就为假时，循环体执行 0 次。

关键

循环体里必须改变条件用到的变量，否则条件永远为真。

while 与 for 的互相改写

for 版本

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

while 版本

```
int i = 1;
while (i <= n) {
    sum += i;
    i++;
}
```

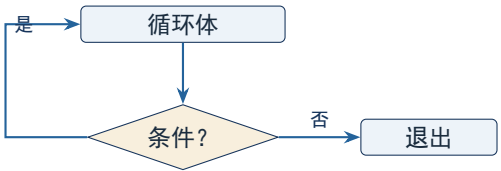
- for 把“初值、条件、更新”集中在一行，适合次数清楚或计数规律明显的循环。
- while 把更新写在循环体里，适合次数未知、靠状态决定是否继续的循环。
- 两者可以互相改写；选择更好读的那一种。

建议

计数用 for，读到结束或满足条件停用 while。

do-while：先执行一次

```
int x;
do {
    cin >> x;
    sum += x;
} while (x != 0);    // 先读再判断
```



- 先执行循环体，再判断条件。
- 循环体至少执行 1 次。

适用

至少要读一个数、至少要执行一次操作时，用 `do-while` 最直接。

while 与 do-while 的区别

	while	do-while
条件位置	循环体之前	循环体之后
最少执行次数	0	1
条件为假时	直接跳过	仍执行一次
常见用途	次数未知的重复	至少做一次的操作

```cpp
int i = 0;
while (i < 0) {
    cout << i;          // 一次也不输出
}
do {
    cout << i;          // 输出一次 0
} while (i < 0);
```

判断

先问“这件事有没有可能一次都不做?”可能，用 `while`；不可能，才考虑 `do-while`。

读到输入结束：while (cin >> x)




```
int x;
long long sum = 0;
int cnt = 0;
while (cin >> x) {
    sum += x;
    cnt++;
}
cout << sum << ' ' << cnt << '\n';
```

- 输入流像一条队列：每执行一次 `cin >> x` 就取走一个数，取不到时循环结束。

不要写

用 `eof()` 判断结束容易多读一次；直接用 `cin >> x` 更稳。

哨兵值：读到 0 停止

```
int x;
int cnt = 0;
cin >> x;
while (x != 0) {          // 0 是结束标志
    if (x > 0) {
        cnt++;
    }
    cin >> x;             // 读下一个数
}
cout << cnt << '\n';
```

- 哨兵值只负责“叫停”，不参与统计。
- 也可以写成 `while (cin >> x && x != 0)`，利用短路：先读成功，再判断值。

顺序

若把 `x != 0` 写在前面，`x` 还没读入，判断没有意义。

while 的常见错误

- 忘记更新：`while (i <= n) { sum += i; }` 中 i 不变，死循环。
- 多写分号：`while (i <= n);` 让循环体变成空语句。
- 条件写反：该继续时写成停止条件，循环一次也不执行。
- 用 `eof()` 判断结束：可能多读最后一次。

```
int i = 1;
while (i <= n); {           // 分号让循环体为空
    sum += i;
    i++;
}
```

检查

编译能通过不代表逻辑对；把 $n = 1$ 代进去，逐轮写出变量值。

【例 4.5】第几项｜如何推导

1. 不知道循环次数，改用 `while`，只要 $s \le m$ 就继续。
2. 每轮先增加项数 n，再把新的一项加入 s。
3. 当循环停止时，首次有 $s > m$，这个 n 就是最小答案。

【例 4.5】第几项｜样例与边界

手算过程

$m = 1000$：$1 + \cdots + 44 = 990 \le 1000$，加上 45 后为 $1035 > 1000$，所以答 45。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.5】第几项｜完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    long long m, s = 0;
    cin >> m;
    int n = 0;
    while (s <= m) {
        s += ++n;                // 先加项数，再加入和
    }
    cout << n << '\n';
    return 0;
}
```

【例 4.6】最大公约数｜如何推导

1. 欧几里得算法利用 $\gcd(a, b) = \gcd(b, a \bmod b)$。
2. 每轮保留余数，令旧的 b 成为新的 a。
3. 当 $b = 0$ 时，当前 a 就是最大公约数。

【例 4.6】最大公约数｜样例与边界

手算过程

$(4,6) \to (6,4) \to (4,2) \to (2,0)$，最终答案为 2。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.6】最大公约数 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    cin >> a >> b;
    while (b) {                    // b 为 0 时结束
        int r = a % b;
        a = b;                     // 旧的 b 变成新的 a
        b = r;
    }
    cout << a << '\n';
    return 0;
}
```

【例 4.7】最小 n 值｜如何推导

1. 从 $n = 0, s = 0$ 开始，逐项加入 $1/n$。
2. 只要和还没达到 m，就继续；停下来的第一项就是最小 n。
3. 和涉及小数，使用 `double`；边界要按“达到”判断。

【例 4.7】最小 n 值｜样例与边界

手算过程

$m = 3$：前 10 项和仍小于 3，加上 $1/11$ 后首次达到 3，答案为 11。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.7】最小 n 值 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int m, n = 0;
    cin >> m;
    double s = 0;
    while (s + 1e-12 < m) {      // 未达到 m 就继续
        s += 1.0 / ++n;
    }
    cout << n << '\n';
    return 0;
}
```

【例 4.8】数据统计｜如何推导

1. 输入没有给个数，使用 `while(cin >> x)` 读到输入结束。
2. 每读入一个数，同时更新最小值、最大值、总和和个数。
3. 最后用 sum/count 算平均值，保留三位小数。

【例 4.8】数据统计｜样例与边界

手算过程

输入 $1, 2, 3$：最小值 1，最大值 3，总和 6，个数 3，均值 2.000。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.8】数据统计｜完整 C++ 程序 1/2

```cpp
#include <bits/stdc++.h>
using namespace std;

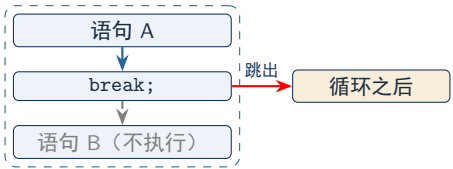
int main() {
    int x, lo = INT_MAX, hi = INT_MIN, n = 0;
    long long sum = 0;
    while (cin >> x) {          // 读到文件结束
        lo = min(lo, x);
        hi = max(hi, x);
        sum += x;
        n++;
    }
```

【例 4.8】数据统计 | 完整 C++ 程序 2/2

```
// 收尾：处理空输入并输出
    if (n > 0) {
        cout << lo << ' ' << hi << ' ' << fixed << setprecision(3);
        cout << (double)sum / n << '\n';
    }
    return 0;
}
```

break：立即跳出当前循环

```
for (int d = 2; d <= n; d++) {
    if (n % d == 0) {
        cout << d << '\n';
        break;          // 找到就停
    }
}
```



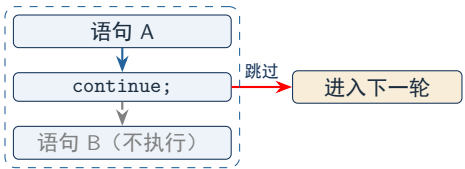
- 执行到 break，整个循环立刻结束。
- 循环后面的语句继续执行。

只跳一层

break 只结束它所在的那一层循环；嵌套时外层照常继续。

continue：跳过本轮剩余语句

```
for (int i = 1; i <= n; i++) {
    if (i % 2 == 0) {
        continue;        // 偶数跳过
    }
    sum += i;            // 只累加奇数
}
```



- continue 不结束循环，只结束本轮。
- 对 for 跳到更新；对 while 跳到条件判断。

易错

在 while 中，若更新语句写在 continue 后面，会被跳过，可能死循环。

break 与 continue 对比

	break	continue
作用	结束整个循环	结束本轮
之后执行	循环后面的语句	下一轮的条件/更新
常见场景	找到答案就停	跳过不满足的数据
嵌套时	只跳最内层	只影响本层本轮

```
for (int i = 1; i <= n; i++) {
    if (i % 3 == 0) {
        continue;       // 跳过 3 的倍数
    }
    sum += i;
}
```

一句话

break 是“不干了”，continue 是“这一个不算，继续下一个”。

常见坑：continue 跳过更新、break 只跳一层

错误：更新被跳过

```
int i = 0;
while (i < 10) {
    if (i % 2 == 0) {
        continue;   // i 没有增加
    }
    i++;
}
```

正确：先更新

```
int i = 0;
while (i < 10) {
    i++;
    if (i % 2 == 0) {
        continue;
    }
    cout << i;
}
```

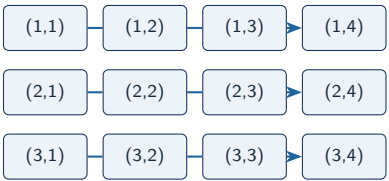
- 死循环的根本原因是“状态没有前进”，不是 continue 本身。
- 嵌套时 break 只跳出最内层；要跳出两层，需在外层再判断或用标志变量。

建议

在 while 里把更新放在 continue 之前，或改用 for。

嵌套循环：外层走一步，内层走一轮

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 4; j++) {
        cout << i << ',' << j << ' ';
    }
    cout << '\n';        // 一行结束
}
```



- 外层 $i = 1$ 时，内层 j 从 1 到 4 完整走一遍。
- 外层再变成 2，内层重新从 1 开始。

顺序

行优先：先固定 i，让 j 跑完；再换下一个 i。总次数 $3 \times 4 = 12$。

嵌套循环的跟踪

外层 i	内层 j	本行输出
1	1,2,3,4	`1,1 1,2 1,3 1,4`
2	1,2,3,4	`2,1 2,2 2,3 2,4`
3	1,2,3,4	`3,1 3,2 3,3 3,4`

- 内层循环在每一轮外层里都要“重新初始化、重新判断、重新更新”。
- 因此两层循环的总次数是外层次数乘以内层次数。
- 手算时先固定外层，完整写出内层的 j 序列，再换下一行。

常见错误

把内层变量在外层之外初始化，导致第二轮内层不执行。

枚举与剪枝：少写一层循环

```
for (int a = 1; a <= 9; a++) {
    int b = 10 - a;          // 由 a+b=10 直接算出 b
    if (b >= 1 && b <= 9) {
        cout << a << ' ' << b << '\n';
    }
}
```

- 枚举：把所有候选逐个试一遍，靠循环实现。
- 剪枝：提前排除不可能的分支，或少枚举一个变量。
- 若 $a + b = 10$，知道 a 就能算出 b，不必再写一层 b 的循环。

代价

枚举的范围越小，程序越快；但剪枝条件必须覆盖所有合法情况。

嵌套循环的常见错误

- 内外层用了同一个变量名：内层会破坏外层的计数。
- 内层循环的花括号漏写：后面的语句被重复执行。
- 把应该在外层做的初始化放进内层：每轮都被重置。
- 输出换行放错位置：所有内容挤成一行，或每输出一个数就换行。

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        cout << i * j << ' ';
    }
    cout << '\n';            // 换行属于外层
}
```

检查

先遮住内层，确认外层逻辑；再遮住外层，确认内层逻辑。

数位拆分模板

```
int x = n;
while (x > 0) {
    int d = x % 10;      // 取个位
    cout << d << ' ';
    x /= 10;             // 删掉个位
}
```

- `% 10` 取最后一位，`/ 10` 去掉最后一位。
- 循环直到 $x = 0$；数字从低位到高位输出。



注意

$n = 0$ 时循环一次也不执行；若题目要求输出 0，要单独处理。

数位统计与反转

```cpp
int x = n;
int cnt = 0, sum = 0, rev = 0;
while (x > 0) {
    int d = x % 10;          // 当前最低位
    cnt++;                   // 位数
    sum += d;                // 数位和
    rev = rev * 10 + d;      // 反转
    x /= 10;
}
cout << cnt << ' ' << sum << ' ' << rev << '\n';
```

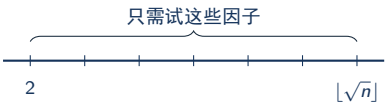
手算 $n = 4728$

$cnt = 4$，$sum = 4 + 7 + 2 + 8 = 21$，rev 依次为 $8 \to 82 \to 827 \to 8274$。

- 四个量同步更新；反转时先乘 10 再加当前位，顺序不能反。

质数判定模板

```
bool prime = true;
if (n < 2) {
    prime = false;       // 0、1 不是质数
}
for (int d = 2; d * d <= n; d++) {
    if (n % d == 0) {
        prime = false;
        break;           // 找到因子就停
    }
}
```



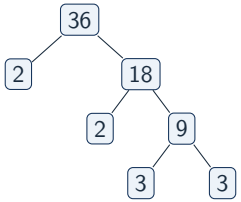
- 若 d 是因子，则 n/d 也是因子；成对出现，只需试到 $\sqrt{n}$。
- 循环结束后 `prime` 仍为真，才是质数。

边界

$n = 2$ 时循环一次也不执行，`prime` 保持真，正确。

质因数分解模板

```
int x = n;
for (int p = 2; p * p <= x; p++) {
    while (x % p == 0) {    // 可能多次
        cout << p << ' ';
        x /= p;
    }
}
if (x > 1) {
    cout << x;              // 剩余因子
}
```



- 从小到大试因子；找到一个就一直除，直到除不尽。
- 循环条件用 `p * p <= x`，因为大的因子成对出现。

输出

36 分解为 $2 \times 2 \times 3 \times 3$。

输出格式：空格与换行

```
bool first = true;
for (int i = 1; i <= n; i++) {
    if (!first) {
        cout << ' ';    // 不是首个
    }
    cout << i;
    first = false;
}
```

1 空格 2 空格 3

行尾没有多余空格

- 用标志变量 first 判断“当前是不是第一个输出”。
- 也可以先输出第一个，再在循环里输出“空格 + 下一个”。

易错

直接在每个数后面输出空格，行尾会多一个空格；有的评测会判格式错误。

调试循环：打印中间状态

```
long long sum = 0;
for (int i = 1; i <= n; i++) {
    sum += i;
    cerr << "i=" << i << " sum=" << sum << '\n';
}
```

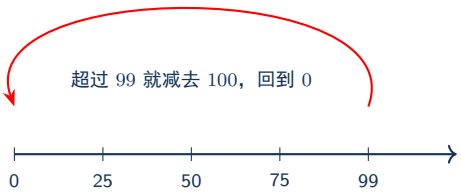
- 在循环体里用 cerr 打印循环变量和关键状态，观察每一轮的变化。
- cerr 不影响标准输出，提交前删掉或注释掉即可。
- 也可以手写跟踪表：列、行分别写轮次和变量值。

方法

先让程序“说话”，再判断是哪一轮、哪个变量开始不对。

为什么只保留余数？

- 只问末两位时，整数 a 与 $a \bmod 100$ 的末两位相同。
- $(a \times b) \bmod 100 = ((a \bmod 100) \times (b \bmod 100)) \bmod 100$。
- 每轮先乘再取余，状态始终小于 100；末六位则改用 10^6。



好处

不必存下 92^{100000} 这样的巨大数，每轮只保存两位，既快又不会溢出。

取余运算的三条性质

性质	例子
$(a+b) \bmod m = (a \bmod m + b \bmod m) \bmod m$	$(17+25) \bmod 10 = 2$
$(a-b) \bmod m = (a \bmod m - b \bmod m + m) \bmod m$	负数要先加 m
$(a \times b) \bmod m = (a \bmod m \times b \bmod m) \bmod m$	$(17 \times 25) \bmod 10 = 5$

```
long long mul = 1;
for (int i = 1; i <= n; i++) {
    mul = mul * x % 100;    // 每一步都取余
}
```

- 乘法取余可以拆开，加法、减法同理；除法没有这个性质。
- 减法出现负数时，先加模数再取余，保证结果非负。

用途

快速幂、阶乘末位、大数取模，都靠这条性质把数控制在小范围。

【例 4.10】末两位数 | 如何推导

1. 只关心末两位，就只保存乘积对 100 的余数。
2. $1992 \bmod 100 = 92$，每轮令 $x = (x \times 92) \bmod 100$。
3. 循环 n 次后输出 x；不必存下巨大乘积。

【例 4.10】末两位数｜样例与边界

手算过程

$n = 3$：$92^2 \bmod 100 = 64$，$64 \times 92 \bmod 100 = 88$。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.10】末两位数 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    int x = 1;
    for (int i = 0; i < n; i++) {
        x = x * 92 % 100;       // 只保留末两位
    }
    cout << x << '\n';
    return 0;
}
```

【例 4.11】体操队｜如何推导

1. 人数除以 $2, 3, 4, 5, 6$ 都余 1，可以从小到大枚举。
2. 每个候选人数再检查除以 7 是否余 0。
3. 第一个同时满足全部条件的数就是最少人数。

【例 4.11】体操队｜样例与边界

手算过程

可把条件合并为人数除以 60 余 1；从 $1, 61, 121, \ldots$ 中找到首个能被 7 整除的数。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.11】体操队 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    for (int x = 1; ; x++) {    // 无限枚举，找到后 break
        bool r1 = (x % 2 == 1 && x % 3 == 1 && x % 4 == 1);
        bool r2 = (x % 5 == 1 && x % 6 == 1);
        if (r1 && r2 && x % 7 == 0) {
            cout << x << '\n';
            break;              // 第一个就是最少人数
        }
    }
    return 0;
}
```

两层循环怎样读？

- 外层每走一步，内层都从头到尾执行一次。
- 画图时外层管“行”，内层管“这一行的字符”。
- 枚举两个未知数时，第三个数常能由总数求得，减少一层循环。
- 对每一组候选值，检查所有约束，再决定是否输出。

质数与平方数的边界

- 1 不是质数。检查素数 x，只需试到 $d^2 \le x$。
- 平方数题先枚举平方根，再检查数字形式，避免对所有四位数开平方。
- 质因数分解时，每除掉一个因子，剩余数就变小；重复因子要反复输出。

【例 4.12】阶乘和｜如何推导

1. 令 f 表示当前阶乘，s 表示阶乘之和。
2. 从 $i = 1$ 起，每轮先计算 $f \leftarrow f \times i$，再更新 $s \leftarrow s + f$。
3. 这样不必每一项都从 1 重新乘到 i。

【例 4.12】阶乘和｜样例与边界

手算过程

$n = 3$：阶乘依次 $1, 2, 6$，累加和依次 $1, 3, 9$。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.12】阶乘和 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    long long f = 1, s = 0;
    for (int i = 1; i <= n; i++) {
        f *= i;                 // 当前阶乘
        s += f;                 // 累加阶乘
    }
    cout << s << '\n';
    return 0;
}
```

【例 4.13】三角形｜如何推导

1. 外层循环控制第几行 i，从 1 到 n。
2. 第 i 行恰好输出 i 个星号。
3. 每行输出后换行；$n = 1$ 时只打印一个星号。

【例 4.13】三角形｜样例与边界

手算过程

$n = 3$ 时三行分别为 *、**、***。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.13】三角形 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cout << string(i, '*') << '\n';  // 第 i 行 i 个星号
    }
    return 0;
}
```

【例 4.14】百钱买百鸡｜如何推导

1. 设公鸡、母鸡、小鸡分别为 a, b, c，先有 $a + b + c = 100$。
2. 价钱条件为 $5a + 3b + c/3 = 100$；为避免小数，可乘 3。
3. 枚举 a, b，用 $c = 100 - a - b$ 算第三种数量，再检查条件。

【例 4.14】百钱买百鸡｜样例与边界

手算过程

例如 $a = 0, b = 25, c = 75$：共 100 只，价钱为 $0 + 75 + 25 = 100$。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.14】百钱买百鸡 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    for (int a = 0; a <= 20; a++) {
        for (int b = 0; b <= 33; b++) {
            int c = 100 - a - b;   // 三种鸡的总数
            if (15 * a + 9 * b + c == 300 && c % 3 == 0) {
                cout << a << ' ' << b << ' ' << c << '\n';
            }
        }
    }
    return 0;
}
```

【例 4.15】水仙花数 | 如何推导

1. 枚举每个三位数 x，范围是 100 到 999。
2. 用除法和取余拆出百位、十位、个位 a, b, c。
3. 若 $a^3 + b^3 + c^3 = x$，就输出该数。

【例 4.15】水仙花数 | 样例与边界

手算过程

153 的各位是 $1, 5, 3$，且 $1^3 + 5^3 + 3^3 = 153$，所以要输出。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.15】水仙花数 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    for (int x = 100; x <= 999; x++) {
        int a = x / 100;         // 百位
        int b = x / 10 % 10;     // 十位
        int c = x % 10;          // 个位
        if (a * a * a + b * b * b + c * c * c == x) {
            cout << x << '\n';
        }
    }
    return 0;
}
```

【例 4.16】找素数 | 如何推导

1. 素数必须大于 1，从 $\max(2, a)$ 开始检查。
2. 对每个候选数，只需寻找不超过 $\sqrt{x}$ 的因子。
3. 找不到因子的数才输出；区间端点都包含。

【例 4.16】找素数｜样例与边界

手算过程

区间 $[5, 10]$ 中，$5, 7$ 无非平凡因子；$6, 8, 9, 10$ 为合数。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.16】找素数 | 完整 C++ 程序 1/2

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    cin >> a >> b;
    for (int x = max(2, a); x <= b; x++) {
        bool prime = true;
        for (int d = 2; d * d <= x; d++) {
            if (x % d == 0) {
                prime = false;
                break;          // 找到因子，提前结束
            }
        }
```

【例 4.16】找素数 | 完整 C++ 程序 2/2

```
// 输出素数并结束
        if (prime) {
            cout << x << '\n';
        }
    }
    return 0;
}
```

【例 4.17】四位完全平方数 | 如何推导

1. 先枚举平方根 x，再计算四位平方数 x^2。
2. 比较千位与百位、十位与个位是否分别相等。
3. 从小到大枚举平方根，输出自然也是升序。

【例 4.17】四位完全平方数｜样例与边界

手算过程

比如 $7744 = 88^2$，前两位同为 7、后两位同为 4，因此符合要求。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.17】四位完全平方数 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    for (int x = 32; x * x <= 9999; x++) {
        int n = x * x;          // 四位平方数
        if (n / 1000 == n / 100 % 10 && n / 10 % 10 == n % 10) {
            cout << n << '\n';
        }
    }
    return 0;
}
```

【例 4.18】分解质因数｜如何推导

1. 从最小可能质因子 2 开始试除。
2. 只要当前数还可被 p 整除，就输出 p 并把数除以 p。
3. 最后若还剩大于 1 的数，它自身就是最后一个质因子。

【例 4.18】分解质因数｜样例与边界

手算过程

$36 \to 18 \to 9 \to 3 \to 1$，依次取出 $2, 2, 3, 3$，输出 `36=2*2*3*3`。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.18】分解质因数 | 完整 C++ 程序 1/2

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    cout << n << '=';
    bool first = true;
    for (int p = 2; p * p <= n; p++) {   // 从小到大试除
        while (n % p == 0) {
            if (!first) {
                cout << '*';
            }
            cout << p;
            first = false;
```

【例 4.18】分解质因数 | 完整 C++ 程序 2/2

```
            n /= p;
        }
    }
    if (n > 1) {                        // 剩余的大于 1
        if (!first) {
            cout << '*';
        }
        cout << n;
    }
    cout << '\n';
    return 0;
}
```

【例 4.19】阶乘之和 | 如何推导

1. 只要答案末六位，阶乘与和每轮都对 10^6 取余。
2. 当阶乘末六位变成 0，以后所有阶乘末六位仍是 0。
3. 因此可以提前停止，避免枚举近百万次。

【例 4.19】阶乘之和 | 样例与边界

手算过程

$n = 10$ 时阶乘和为 4037913，末六位为 37913，输出时不补前导零。

动手检查

对照输入与输出，再自行选一个最小值或临界值，逐轮写出变量的变化。

【例 4.19】阶乘之和 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    long long f = 1, s = 0;
    for (int i = 1; i <= n; i++) {
        f = f * i % 1000000;    // 阶乘末六位
        s = (s + f) % 1000000;  // 和的末六位
        if (f == 0) {
            break;              // 后面全是 0，提前结束
        }
    }
    cout << s << '\n';
    return 0;
}
```

练习路线

1. 先做计数、求和、均值、最大最小，练好一层循环。
2. 再做不知道次数的循环、数位处理、取余和精度控制。
3. 最后做图形、质因数、质数与方程枚举。

索引说明

下面列出截图中全部 49 个练习位置；其中 6 个重复出现，实际是 43 道不同题。蓝色链接均逐题核对了学校 OJ 题名与样例。

练习索引 | 位置 1-7 / 49

1. P5059 | 求平均年龄（一本通 1059）
2. P5060 | 均值（一本通 1060）
3. P5061 | 求整数的和与均值（一本通 1061）
4. P5062 | 最高的分数（一本通 1062）
5. P5063 | 最大跨度值（一本通 1063）
6. P5064 | 奥运奖牌计数（一本通 1064）
7. P5065 | 奇数求和（一本通 1065）

练习索引 | 位置 8-14 / 49

8. P5066 | 满足条件的数累加（一本通 1066）
9. P5067 | 整数的个数（一本通 1067）
10. P5068 | 与指定数字相同的数的个数（一本通 1068）
11. P5069 | 乘方计算（一本通 1069）
12. P5070 | 人口增长（一本通 1070）
13. P5071 | 菲波那契数（一本通 1071）
14. P5072 | 鸡尾酒疗法（一本通 1072）

练习索引 | 位置 15-21 / 49

15. P5073 | 救援（一本通 1073）
16. P5074 | 津津的储蓄计划（一本通 1074）
17. P5075 | 药房管理（一本通 1075）
18. P5076 | 正常血压（一本通 1076）
19. P5077 | 统计满足条件的 4 位数（一本通 1077）
20. P5078 | 求分数序列和（一本通 1078）
21. P5079 | 计算分数加减表达式的值（一本通 1079）

练习索引 | 位置 22-28 / 49

22. P5080 | 余数相同问题（一本通 1080）
23. P5081 | 分苹果（一本通 1081）
24. P5082 | 求小数的某一位（一本通 1082）
25. P5083 | 计算星期几（一本通 1083）
26. P5084 | 幂的末尾（一本通 1084）
27. P5085 | 球弹跳高度的计算（一本通 1085）
28. P5086 | 角谷猜想（一本通 1086）

练习索引 | 位置 29-35 / 49

29. P5087 | 级数求和（一本通 1087）
30. P5088 | 分离整数的各个数（一本通 1088）
31. P5089 | 数字反转（一本通 1089）
32. P5090 | 含 k 个 3 的数（一本通 1090）
33. P5085 | 球弹跳高度的计算（一本通 1085）
34. P5086 | 角谷猜想（一本通 1086）
35. P5087 | 级数求和（一本通 1087）

练习索引 | 位置 36-42 / 49

36. P5088 | 分离整数的各个数（一本通 1088）
37. P5089 | 数字反转（一本通 1089）
38. P5090 | 含 k 个 3 的数（一本通 1090）
39. P5091 | 求阶乘的和（一本通 1091）
40. P5092 | 求出 e 的值（一本通 1092）
41. P5093 | 计算多项式的值（一本通 1093）
42. P5094 | 与 7 无关的数（一本通 1094）

练习索引 | 位置 43-49 / 49

43. P5095 | 数 1 的个数（一本通 1095）
44. P5096 | 数字统计（一本通 1096）
45. P5097 | 画矩形（一本通 1097）
46. P5098 | 质因数分解（一本通 1098）
47. P5099 | 第 n 小的质数（一本通 1099）
48. P5100 | 金币（一本通 1100）
49. P5101 | 不定方程求解（一本通 1101）