04 循环

从状态变化理解循环 · 3 学时

本讲学习目标

- 追踪短循环的每轮判断、状态与输出
  - 说明循环体执行次数、退出依据与最终状态
- 独立编写简单单层计数或累加循环
  - 用正常与边界情况核查结果
- 在给定结构中追踪、补全两层固定次数循环
  - 解释一行与行内各列怎样形成
- 识别 do while 的后判断特点，诊断并修正短循环中的条件或更新错误
  - 解释首次执行差异及修正后的正常与边界行为

循环：从重复操作到迭代自动化

程序 = 数据对象 + 操作 + 控制

- 循环实现**迭代自动化**，使程序按照规则重复执行操作。
- 循环实现**操作规模突破**，大量重复操作不再需要逐项编写代码。

1 从重复任务到条件控制

由逐句输出进入有进度、条件和更新的重复处理。

逐句输出 1 ~ 5

```c
#include <stdio.h>
int main(void) {
    printf("1\n");
    printf("2\n");
    printf("3\n");
    printf("4\n");
    printf("5\n");
    return 0;
}
```

逐行输出

五条 printf 依次执行，各输出一行

第 1 行：1
第 2 行：2
第 3 行：3
第 4 行：4
第 5 行：5

数字与次数

每个数字和输出次数都写在代码中

输出范围扩大后的重复处理

- 同一输出任务的上界改变
  - 原任务：逐行输出 1 ~ 5
  - 新任务：逐行输出 1 ~ 100
- 找出随上界变化的内容和保持不变的输出动作
- 提出避免逐句写出 100 条输出语句的办法

while 的条件、循环体与更新

```c
#include <stdio.h>
int main(void) {
    int i = 1;
    while (i <= 5) {
        printf("%d\n", i);
        i++;
    }
    printf("exit i = %d\n", i);
    return 0;
}
```

进度与继续条件

i 表示下一次要输出的数字，从 1 开始

条件 i <= 5 使用当前 i，判断是否进入循环体

循环体中的处理与更新

本例的复合语句是**循环体**，由 while 后的花括号包围

printf 输出当前 i，随后 i++ 将 i 加 1

循环退出与上界变化

条件为假后，才执行循环后的退出状态输出

改为输出 1 ~ 100 时，循环体不变，继续条件改为 i <= 100

这种按规则反复执行操作称为**迭代自动化**；操作次数增加，不必逐项增加输出语句

练习：while 的逐轮执行

```
1 int i = 1;
2 while (i <= 5) {
3     printf("%d\n", i);
4     i++;
5 }
6 printf("exit i = %d\n", i);
```

判断前 i	条件真假	循环体输出	更新后 i
?	?	?	?
?	?	?	?
…	…	…	…

从 i=1 开始，按“判断条件 → 为真时输出 → 更新 → 再次判断”的顺序分析

每次判断填写一行，直到条件首次为假；条件为假时，不执行本轮循环体，也不更新 i

按实际判断次数继续填写表格，保留最后一次假判断

给出循环体执行次数、条件求值次数和退出时的 i

解析：while 的逐轮执行

判断前 i	条件真假	循环体输出	更新后 i
1	真	1	2
2	真	2	3
3	真	3	4
4	真	4	5
5	真	5	6
6	假	无	无本轮更新；退出时 i=6

循环体执行 5 次；第 6 次条件求值为假，退出时 i=6

退出循环后，继续执行循环后的 printf，输出 exit i = 6

while 的语法与执行过程

```
while (expression)
    语句
```

```
while (expression) {
    /* 循环体 */
}
```

左：while 的一般形式；语句表示循环体，是占位说明

右：多条语句常用 {} 包围，构成复合语句

expression 是**循环条件**表达式

每次先计算条件：非 0 为真，执行循环体；0 为假，退出

循环体完整执行一次称为一次**迭代**；结束后回到条件判断

更新由实际代码完成，while 不自动改变变量

初值变化后的执行次数

i 的初值	第一次判断	循环体执行次数	退出 i
6	?	?	?
5	?	?	?
1	?	?	?

保持前面循环的条件和循环体不变，只改变 i 的初值

先判断循环的零次执行边界

i 的初值	第一次判断	循环体执行次数	退出 i	循环体输出
6	6 <= 5 为假	0	6	无
5	5 <= 5 为真	1	6	5
1	1 <= 5 为真	5	6	1 ~ 5

while 在进入循环体之前判断；首次为假时循环体执行零次

三种初值最终都得到 i=6，循环后的 printf 都输出 exit i = 6

2 for 的控制与执行顺序

由 while 的先判断模型进入 for 的语法、控制和相互转换。

for 头部的起点、条件与更新

```c
#include <stdio.h>
int main(void) {
    int i;
    for (i = 1; i <= 5; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

头部的控制信息

for 头部写出起点、条件和更新

- i = 1 设置起始值，只执行一次
- i <= 5 在每次执行循环体前判断
- i++ 在每次循环体执行结束后，将 i 加 1

循环体与首轮

循环体逐行输出当前 i

第一轮输出 1 后，i 更新为 2

末轮与退出

第五轮输出 5 后，i 更新为 6；再次判断条件为假，退出循环

for 的语法与执行顺序

```
for (expression1; expression2; expression3) {
    /* 循环体 */
}
```

头部的三个表达式

两个分号分隔三部分；expression1 ~ 3 是占位符

expression1：首次判断前执行 i = 1 一次

expression2：每轮循环体前计算 i <= 5

结果非 0 为真，0 为假

expression3：每轮循环体结束后执行 i++

执行顺序

初始化 → 判断 → 为真时执行循环体 → 更新 → 再判断

条件为假时退出，不执行本轮循环体和更新

循环体的形式

一条语句，或 {} 包围的复合语句

控制输出范围与间隔

```c
int i;
for (i = 2; i <= 10; i += 2) {
    printf("%d\n", i);
}
```

目标序列

任务：逐行输出 2、4、6、8、10

起点、条件与更新

起点 i = 2：从第一个要输出的数字 2 开始

条件 i <= 10：当前数字不超过 10 时，执行循环体

更新 i += 2：每次输出后将 i 加 2，得到下一个偶数

末轮与控制依据

输出 10 后，更新得到 i = 12；下一次判断为假，退出循环

编写循环时，需要确定起始值、继续条件和每次更新的方式

练习：for 的逐轮执行

```c
int i;
for (i = 2; i <= 4; i++) {
    printf("%d\n", i);
}
```

判断前 i	i <= 4	循环体输出	更新后 i
?	?	?	?
?	?	?	?
…	…	…	…

写出每次判断前的 i、条件真假、循环体输出与更新后的 i

每次判断填写一行，直到条件首次为假；最后一次假判断也要填入表中

给出循环体执行次数、条件求值次数及退出时的 i

解析：for 的逐轮执行

判断前 i	i <= 4	循环体输出	更新后 i
2	真	2	3
3	真	3	4
4	真	4	5
5	假	无	无本轮更新

循环体执行 3 次，条件求值 4 次，退出时 i = 5

计数控制与处理状态控制

```c
int i;
for (i = 1; i <= 5; i++) {
    printf("%d\n", i);
}
```

```c
int n;
for (n = 123; n > 0; n /= 10) {
    printf("%d\n", n);
}
```

左：**计数控制**。先确定目标次数，用 i 记录进度，输出 5 次

右：**按处理状态控制**。检查待处理数据，用 n > 0 决定是否继续

整数除法 n /= 10 使 n 依次变为 12、1、0

输出剩余整数 123、12、1，不是单个数位；没有另设目标次数

两者都依据条件判断；计数变量也是程序状态的一部分

区分依据是条件检查计数进度还是待处理数据

计数目标可在运行时给定；状态控制的次数也可能由数据推算

逗号表达式在 for 头部的用法

```c
int i, j;
for (i = 1, j = 5; i <= 5 && j > 0; i++, j--) {
    printf("%d %d\n", i, j);
}
```

逗号表达式用逗号运算符连接多个表达式

- 各个表达式从左到右依次求值
- 整个表达式的值取最后一个表达式的值

示例每轮输出当前的 i 和 j

- 初始化时先执行 i = 1，再执行 j = 5
- 每轮输出后先执行 i++，再执行 j--

位置	表达式	作用
初始化	i = 1, j = 5	依次设置两个初值，整个初始化表达式只执行一次
条件	i <= 5, j > 0	两项依次求值，仅用最后一项 j > 0 的结果决定是否继续
更新	i++, j--	每次循环体结束后，依次执行两个更新

示例代码用 i <= 5 && j > 0 要求两个条件都成立

- 条件中的逗号只取最后一项的值，不表示“两个条件同时满足”

两种逗号的作用不同

- int i, j; 中的逗号是声明分隔符
- 头部初始化和更新中的逗号是逗号运算符

两个分号仍分隔初始化、条件和更新

for 与 while 的相互转换

```c
int i;
for (i = 1; i <= 5; i++) {
    printf("%d\n", i);
}
```

```c
int i;
i = 1;
while (i <= 5) {
    printf("%d\n", i);
    i++;
}
```

左 for，右 while；由 for 改写为 while：

初始化 i = 1 移到循环前，仍只执行一次

条件 i <= 5 仍在循环体之前判断

更新 i++ 移到循环体末尾，仍在输出后执行

反向改写：初始化、条件、末尾更新移回 for 头部，循环体保留输出

本例都输出 1～5，循环体执行 5 次、判断 6 次，退出 i = 6

3 do while 的后判断

改变判断位置，比较首次执行，再回顾三种循环形式。

do while：先执行循环体，再判断

```c
#include <stdio.h>
int main(void) {
    int i = 1;
    do {
        printf("%d\n", i);
        i++;
    } while (i <= 5);
    return 0;
}
```

首轮与轮后判断

第一轮：输出 1 → i++ 将 i 更新为 2 → 判断 2 <= 5 为真，继续循环

循环体执行结束后，判断 i <= 5

条件为真：返回循环体，执行下一轮

条件为假：退出循环，执行循环后的语句

末轮与运行结果

最后一轮：输出 5 → i 更新为 6 → 判断 6 <= 5 为假，退出循环

本例每行输出一个整数，依次为 1～5；退出时 i = 6

后判断循环

do while 是先执行循环体、再判断条件的循环形式，称为**后判断循环**

do while 的语法与执行顺序

```
do {
    /* 循环体 */
} while (expression);
```

循环体与轮后条件

expression 是条件表达式的占位符；前页对应 i <= 5

循环体可为一条语句，也可为 {} 包围的复合语句

执行与判断的顺序

循环体 → 计算条件 → 决定是否重复

非 0（真）：再次执行循环体；0（假）：退出，执行循环后的语句

每次进入 do while，先执行循环体，再判断；即使初始状态不满足条件，也会执行一次

末尾的分号

while (expression); 的末尾分号是完整 do while 语句的必要部分

练习：条件为假时的首次执行

```c
while (0) {
    printf("X\n");
}
```

```c
do {
    printf("X\n");
} while (0);
```

条件都为 0，循环体都只输出一行 X

按各自的执行顺序填写下表：输出什么、循环体执行几次、在哪里判断条件

循环形式	输出	循环体执行次数	判断位置
while	?	?	?
do while	?	?	?

解析：条件为假时的首次执行

循环形式	输出	循环体执行次数	判断位置
while	无	0	循环体之前
do while	一行 X	1	循环体之后

while：先判断 0 为假，直接跳过循环体

do while：先输出 X，再判断 0 为假，退出循环

两段代码的条件相同，首次执行差异来自判断位置

for 与 do while 的相互转换

```c
int i;
for (i = 1; i <= 5; i++) {
    printf("%d\n", i);
}
```

```c
int i = 1;
do {
    printf("%d\n", i);
    i++;
} while (i <= 5);
```

左 for，右 do while；本例可按以下位置双向改写：

初始化：for 头部第一部分 ↔ do while 循环前

输出：均在循环体内；更新：头部第三部分 ↔ 循环体末尾

条件：循环体前 ↔ 循环体后；do while 末尾保留分号

初值为 1，首次条件为真：都输出 1 ~ 5，执行 5 次，退出 i = 6

for 判断 6 次，do while 判断 5 次：for 多一次首次输出前的判断

若初值为 6：for 不输出，do while 先输出 6，再更新并判断

因此转换前须确认是否应执行第一轮，不能只移动代码位置

三种循环形式的比较

循环形式	判断位置	循环体最少执行次数	更新位置
while	循环体之前	0	本讲示例在循环体内更新
for	循环体之前	0	本讲示例在头部第三部分更新
do while	循环体之后	1	本例在循环体内更新

while 和 do while 的更新由循环体中的代码完成；for 头部第三部分在每次循环体结束后执行

选择形式时，先看任务是否需要在首次判断之前完成一次处理

先判断是否需要处理：采用 while 或 for

先处理一次，再判断是否继续：采用 do while

while 和 for 都能用于计数控制和处理状态控制

起始值、条件和更新适合集中写在头部时，可用 for

主要围绕一个继续处理的条件组织代码时，可用 while

4 计数进度与累加结果

从输出进入累加，分清下一项和已得结果；应用已学多初值初始化。

用 count 和 sum 完成求和

- 任务：用循环计算 1 + 2 + 3 + 4 + 5
- 用两个变量分别保存下一项和当前总和
  - count = 1：下一项从 1 开始
  - sum = 0：还没有加入任何一项，当前总和为 0
- 每轮先将当前 count 加入 sum，再将 count 加 1，准备处理下一项
- 两个变量的值共同表示求和进行到哪里、已经得到多少，这就是本例需要保存的两种状态

求和循环：先累加，再更新 count

```c
#include <stdio.h>
int main(void) {
    int count = 1;
    int sum = 0;
    while (count <= 5) {
        sum += count;
        count++;
    }
    printf("sum = %d, count = %d\n",
           sum, count);
    return 0;
}
```

继续条件

count <= 5：还有需要相加的项时，继续循环

累加与推进

sum += count; 累加当前项，与 sum = sum + count; 等价

count++;：将 count 加 1，准备下一项，并影响下一次条件判断

顺序是先累加、再更新 count；若先执行 count++，第一次加入的就会是 2

循环后的输出

循环结束后，printf 输出最终的 sum 和 count

逐轮累加的过程

本轮相加的 count	累加前 sum	累加后 sum	更新后 count
1	0	1	2
2	1	3	3
3	3	6	4
4	6	10	5
5	10	15	6

每轮从上一轮留下的 sum 和 count 开始

以第三轮为例：sum = 3 + 3 = 6，随后 count 从 3 更新为 4

第五轮得到 sum = 15，count 更新为 6；再次判断 6 <= 5 为假，退出循环，不再累加

最终输出：sum = 15, count = 6

count 与 sum 的不同作用

比较项	count：下一项	sum：当前总和
初值	1，从第一项开始	0，尚未加入任何一项
每轮更新	count++，准备下一项	sum += count，加入当前项
是否决定继续循环	通过 count <= 5 决定	不参与本例的循环条件
退出值	6，下一项超过上界 5	15，保存所求的总和

本例中，count 的值表示下一项，承担**进度状态**的作用

sum 的值表示已累计的和，承担**结果状态**的作用

每轮先更新 sum，再更新 count，才能将当前这一项加入总和

分析其他累加任务时，分别确定：下一项怎样变化、何时停止，以及结果从什么值开始、每轮怎样累加

练习：读懂 for 的两个初值

```c
#include <stdio.h>
int main(void) {
    int count, sum;
    for (count = 1, sum = 0;
         count <= 5;
         count++) {
        sum += count;
    }
    printf("sum = %d, count = %d\n",
           sum, count);
    return 0;
}
```

两处逗号

找出声明与 for 初始化部分的两处逗号，说明各自的作用

初始化的执行

写出首次判断前 count、sum 的赋值顺序，以及各项赋值执行几次

顺序与结果

与前面的 while 程序比较：每轮相加和更新的顺序是否相同？写出退出时 count 和 sum 的值

解析：for 的初始化与求和结果

```c
#include <stdio.h>
int main(void) {
    int count, sum;
    for (count = 1, sum = 0;
         count <= 5;
         count++) {
        sum += count;
    }
    printf("sum = %d, count = %d\n",
           sum, count);
    return 0;
}
```

声明中的逗号

int count, sum; 声明两个 int 变量，逗号分隔声明中的两个变量名，不是逗号运算符

初始化中的逗号

for 初始化部分的逗号运算符连接两个赋值表达式

先执行 count = 1，再执行 sum = 0；两项赋值各执行一次

初始化在首次判断前完成，后续轮次不会重新将 sum 设为 0，因此累计结果可以保留下来

每轮顺序与结果

每轮先在循环体内执行 sum += count;，再执行头部的 count++

相加和更新的顺序与 while 程序相同；退出时 sum = 15、count = 6

练习：补全求和循环

```
1 int limit = 5, count = 1, sum = 0;
2 while (__________) {
3     sum += count;
4     __________;
5 }
```

任务范围

limit 表示最后一个要相加的整数；本例从 1 开始，包含 limit 本身。若 limit 小于 1，则没有需要相加的项

补全与正常结果

补全继续条件和 count 的更新语句

limit 为 5 时，写出最终的 sum

零次执行的检查

只将初值 limit 改为 0，count 和 sum 仍分别从 1、0 开始：写出循环体执行次数及退出时的 sum、count

解析：求和循环与零次执行

```
1 while (count <= limit) {
2     sum += count;
3     count++;
4 }
```

条件与更新

count <= limit 时才累加，保证上界这一项也能加入总和

count++ 在累加后准备下一项；超过上界后，下一次判断结束循环

正常上界

limit=5：五轮加入 1～5，得到 sum=15 和 count=6

下一次判断为假，循环结束

零次边界

limit = 0：首次判断 1 <= 0 为假，循环体执行零次

两个变量都保留初值：sum = 0、count = 1

5 非负整数的逐位取数

用商与余数追踪待处理部分，以整数 0 的边界应用后判断。

用 while 逐位取出数字

```c
#include <stdio.h>
int main(void) {
    int n = 10005;
    while (n > 0) {
        int digit = n % 10;
        printf("%d\n", digit);
        n = n / 10;
    }
    return 0;
}
```

取位与剩余部分

每轮先取当前最低位，再把剩余高位写回 n

条件依据尚待处理的 n 判断；输出顺序从低位到高位

首轮与下一轮

第一轮取出数位 5，剩余 1000

10005 % 10 = 5，10005 / 10 = 1000

下一轮用新的 n=1000 判断和取位

练习：逐位处理 10005

判断前 n	本轮余数	更新后 n
10005	5	1000
1000	?	?
100	?	?
10	?	?
1	?	?

沿用前页的 n % 10 和 n / 10，补全各轮并指出下一次判断何时为假

解析：10005 的各位数字与退出

判断前 n	本轮余数	更新后 n
10005	5	1000
1000	0	100
100	0	10
10	0	1
1	1	0

完整过程：依次输出 5、0、0、0、1，随后以 n=0 退出

至少取出一位

```c
#include <stdio.h>
int main(void) {
    int n = 0;
    do {
        int digit = n % 10;
        printf("%d\n", digit);
        n = n / 10;
    } while (n > 0);
    return 0;
}
```

初值为 0 的问题

整数 0 有一位数字 0，但原 while (n > 0) 在初值 n = 0 时不执行，无法输出这一位

先处理，再判断

使用 do while：先取位、输出并更新 n，再用 n > 0 判断是否继续

两种初值的结果

初值 n=0：先输出一位 0，更新后 n 仍为 0，条件为假，循环结束

初值改为 n=10005：依次输出 5、0、0、0、1，随后以 n=0 退出

6 两层循环形成行列

区分一行与行内一个字符两种迭代单位。

两层循环的行列分工

行	目标输出
第 1 行	****
第 2 行	****
第 3 行	****

外层循环每轮完成一行；**内层循环**每轮输出一个 *，一行输出四个后换行

外层循环体中包含内层循环，称为两层**循环嵌套**

完成第一行后，外层进入第二行；再次执行内层循环时，从第一列开始处理

两层 for 的执行过程

```c
#include <stdio.h>
int main(void) {
    int row, col;
    for (row = 1; row <= 3; row++) {
        for (col = 1; col <= 4; col++) {
            putchar('*');
        }
        putchar('\n');
    }
    return 0;
}
```

每行的列初值与输出

内层 for 位于外层循环体内，每次进入都先执行 col = 1

第一行：row = 1，col 从 1 到 4，每次输出一个 *

内层退出与换行

col 更新为 5 后，col <= 4 为假，内层结束；随后执行 putchar('\n'); 换行

换行语句在内层之后、外层循环体内，因此每行完成后换行一次

外层推进与总输出

再执行 row++，进入下一次外层判断；程序共输出三行，每行四个星号

练习：第二行的起点与星号总数

```c
#include <stdio.h>
int main(void) {
    int row, col;
    for (row = 1; row <= 3; row++) {
        for (col = 1; col <= 4; col++) {
            putchar('*');
        }
        putchar('\n');
    }
    return 0;
}
```

当前执行位置

当前 row=2，即将再次执行内层 for

起点与数量

给出第二行第一次判断前的 col、该行输出的字符数和三行的字符总数

解析：每行从 col = 1 开始

行 row	内层 col	本行输出
1	1 ~ 4	****
2	1 ~ 4	****
3	1 ~ 4	****

每次外层执行到内层头部，col=1 再次建立

三行共输出 3×4=12 个 *

练习：补全两行三列

```
for (int row = 1; row <= ___; row++) {
    for (int col = 1; col <= ___; col++) {
        putchar('*');
    }
}
```

目标输出

目标：两行，每行三个星号

补全与检查

填写两个上界，在代码中添加换行语句，并给出总字符数

解析：两行三列的输出

```
1 for (int row = 1; row <= 2; row++) {
2     for (int col = 1; col <= 3; col++) {
3         putchar('*');
4     }
5     putchar('\n');
6 }
```

内外层的执行次数

外层两次；每次内层三次，内层初值每行重新建立

输出结果

输出两行，每行三个星号，共六个星号

7 常见循环错误

沿实际执行找首次偏差，再做局部修正与边界复验。

漏掉最后一项

```c
int i = 1;
while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

目标与当前条件

目标：逐行输出 1～5；当前代码使用条件 i < 5

偏差与依据

实际只输出 1～4

i=1、2、3、4 时条件为真，依次输出 1～4

到 i=5 时，5<5 为假，首次偏离“还应输出 5”的目标

修正与验证

将条件改为 i<=5 后，输出 1～5，退出 i=6

缺少更新

```c
int i = 1;
while (i <= 5) {
    printf("%d\n", i);
    /* 此处没有 i++ */
}
```

重复不变的状态

静态追踪：前几次判断前都是 i=1，条件反复为真，输出反复为 1

修正更新位置

修正：在输出之后加入 i++，让下一次判断使用新的 i

修正后的输出与退出

修正后输出 1～5；输出 5 后 i=6，再判断 6<=5 为假，循环结束

多余分号与空循环体

```c
int i = 1;
while (i <= 5);
{
    printf("%d\n", i);
    i++;
}
```

循环体的归属

分号构成 while 的**空循环体**

后面的复合语句不属于该循环；缩进不能把它变成循环体

初值 i=1 时条件反复为真，循环体不更新 i；后面的输出语句无法到达

修正与验证

删除多余分号后，输出与更新才成为循环体内容；依次输出 1～5，退出时 i = 6

练习：修正循环错误

A

```c
int i = 2;
while (i < 4) {
    printf("%d\n", i);
    i++;
}
```

B

```c
int i = 2;
while (i <= 4) {
    printf("%d\n", i);
}
```

目标：逐行输出 2、3、4。分别说明 A、B 实际怎样执行，找出最早与目标不符的位置，并修改代码

检查修正版：保留其余代码，分别将初值改为 i = 4、i = 5，写出输出和循环体执行次数

按“比较目标与实际执行 → 找出首次不符的位置 → 修正并检查结果”的顺序作答

B 原版只在纸上分析，不直接运行

解析：错误原因与修正

- A：i=2、i=3 时输出；到 i=4，4<4 为假，首次遗漏目标中的 4
  - 将条件改为 i<=4；从 2 开始输出 2、3、4，退出时 i=5
- B：首次输出 2 后 i 仍为 2，下一次判断仍真，反复输出 2
  - 循环体内补 i++；从 2 开始输出 2、3、4，退出时 i=5
- 两种修正版从 i = 4 开始各输出一次 4，更新为 5 后退出；从 i = 5 开始均不输出，循环体执行零次
- 检查方法：沿初值、条件、循环体中的输出与更新逐步比较目标和实际执行，找出首次不符的位置；修正后检查零次、一次和多次执行的结果
- 还要确认循环体归属，以及更新后的值能否使条件转假；仅有更新语句不能保证循环结束

本讲总结

- 循环实现迭代自动化，突破逐项编写重复操作的限制
  - while、for 先判断：条件为真时执行循环体并按代码更新状态，条件为假时退出
  - 输出 1～100 仍使用同一循环体；逐位取数按运行时剩余数值决定何时停止
  - 循环体次数、条件求值次数和退出状态分别判断，重复执行不只用于固定次数
- while 与 for 可以完成相同的重复任务
  - 根据起点、条件和更新在代码中的组织方式选择更清楚的写法
  - do while 先执行循环体再判断，首次条件为假时仍执行一次
- 不同任务需要不同的状态
  - 计数表示处理进度，累加值保存结果，逐位取数中的整数表示尚待处理的部分
  - 两层循环分别控制行与行内字符；每次进入内层头部时重新建立列初值
- 构造与检查循环时，对照目标核查初值、条件、循环体、更新和退出
  - 用正常与边界情形发现末项遗漏、更新缺失或循环体位置错误

本讲内容与教材的对应

《C Primer Plus》第六版	本讲阅读要求
第 6 章 6.1～6.3 节	while 的条件、执行与退出
第 6 章 6.5、6.7～6.9 节	for 的顺序与逗号初始化；do while 的后判断；循环形式选择
第 6 章 6.10 节	两层循环及内层的重新初始化

10005 逐位取数是本课程应用例子。本讲不要求独立构造复杂的 do while、两层循环或数组程序。