Never be tired to repeat.

C程序设计

T06

C控制语句：循环

厦门大学信息学院软件工程系

黄炜 副教授

主要内容

- 循环的要素
- 循环的流程
- 关系运算符
- 循环的灵活用法
- 循环的中途退出
- 数组初步

1. 循环的要素

循环的三个要素

- 一般的循环具有三个要素
  - 初始状态的设定
  - 循环的条件
  - 循环状态的改变
- 特殊的循环
  - 死循环：循环条件永远为真，无法停止
  - 无意义的循环：循环条件永远为假，无法进入循环

三种类型的循环

- 三种类型的循环
  - do-while 循环的循环体至少执行一次

选用何种循环应根据实际而定，使得代码简单

```
[初始状态;]
while (<循环条件表达式>)
{
    [循环语句（含状态的改变）]
}
```

```
[初始状态;]
do
{
    [循环语句（含状态的改变）]
}
while (<循环条件表达式>);
```

```
for ([初始状态]; [循环条件表达式]; [循环语句（状态的改变）])
{
    [循环语句（可包含状态的改变）]
}
```

```c
/* summing.c -- sums integers entered interactively */
#include <stdio.h>
int main(void)
{
    long num;
    long sum = 0L;      /* initialize sum to zero   */
    int status;

    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    status = scanf("%ld", &num);
    while (status == 1) /* == means "is equal to"   */
    {
        sum = sum + num;
        printf("Please enter next integer (q to quit): ");
        status = scanf("%ld", &num);
    }
```

初始状态

循环条件表达式
（==表示判断是否相等）

状态改变

```
    printf("Those integers sum to %ld.\n", sum);

    return 0;
}
```

```
Please enter an integer to be summed (q to quit): 55↲
Please enter next integer (q to quit): 33↲
Please enter next integer (q to quit): 235↲
Please enter next integer (q to quit): q↲
Those integers sum to 323.
```

```c
// when.c -- when a loop quits
#include <stdio.h>
int main(void)
{
    int n = 5;

    while (n < 7)                       // line 7
    {
        printf("n = %d\n", n);
        n++;                            // line 10
        printf("Now n = %d\n", n);      // line 11
    }
    printf("The loop has finished.\n");

    return 0;
}
```

```
n = 5
Now n = 6
n = 6
Now n = 7
The loop has finished.
```

2. 循环的流程

循环的流程

- 循环符合条件则进循环体；否则跳过语句体。

```
1 始状态;]
while 2 循 4 条 6 达式>)
{
   3 循 5 语句（含状态的改变）]
}
7 环体之外的语句]
```

```
1 始状态;]
while 2 循环条件表达式>)
{
      [循环语句（含状态的改变）]
}
3 环体之外的语句]
```

```
1 始状态;]
do
{
    2 环 4 句（含状态的改变）]
}
while 3 循 5 条件表达式>);
6 环体之外的语句]
```

```
1 始状态;]
do
{
    2 环语句（含状态的改变）]
}
while (3 环条件表达式>);
4 环体之外的语句]
```

循环的流程

- 循环符合条件则进循环体；否则跳过语句体。

```
for (1 始状态]; [2 不5 8 式]; [4 不7 （状态的改变）])
{
    [3 不6 （可包含状态的改变）]
}
9 环体之外的语句]
```

```
for (1 始状态]; [2 不条件表达式]; [循环语句（状态的改变）])
{
    [循环语句（可包含状态的改变）]
}
3 环体之外的语句]
```

循环的流程

- 循环的终止
  - 符合条件则进入循环语句体；否则跳过语句体。

死循环（永不停止）	循环3次

`index = 1;`
`while (index < 5)`
`    printf("%d\n", index);`
状态不改变
	`index = 1;`
	`while (++index < 5)`
	`    printf("%d\n", index);`

循环$2^{31}+1$次	**不循环（循环0次）**

`index = 1;`
`while (--index < 5)`
`    printf("%d\n", index);`
	`index = 10;`
	`while (++index < 5)`
	`    printf("%d\n", index);`

  - 终止条件往往是出错的关键部位

```c
/* while1.c -- watch your braces        */
/* bad coding creates an infinite loop */
#include <stdio.h>
int main(void)
{
    int n = 0;

    while (n < 3)
        printf("n is %d\n", n);
        n++;
    printf("That's all this program does\n");

    return 0;
}
```

```
n is 0
n is 0
n is 0
n is 0
（永不停止）
```

C程序中，空白字符（空格、制表符、回车）的意义是相同的。不能用缩进来判断循环体的范围。

应使用正确的缩进规范，便于读者理解

循环的语句体范围

- 循环的语句体范围
  - 循环的内容是在while或for之后最近的（或do-while之间的）一个语句体或一条语句，用花括号分隔，缩进一层。

```
while (a<5)
{
    a++;
    printf("%d",a);
}
```

```
for (i=0; i<5; i++)
{
    a++;
    printf("%d",a);
}
```

```
do
{
    a++;
    printf("%d",a);
}
while (a<5);
```

```
while (a<5)
    a++;
printf("%d",a);
```

```
for (i=0; i<5; i++)
    a++;
printf("%d",a);
```

循环的语句体范围

- 循环的语句体范围
  - 循环的内容是在while或for之后最近的（或do-while之间的）一个语句体或一条语句，用花括号分隔，缩进一层。
  - 如果循环的内容是一条语句，可以不用花括号分隔。
    - 注意不要多加分号，因为空语句“;”也是一条语句

```
while (a<5)
    a++;
printf("%d",a);
```

```
for (i=0; i<5; i++)
    a++;
printf("%d",a);
```

```
do
    a++;
while (a<5);
printf("%d",a);
```

```
while (a<5);
a++;
```

```
for (i=0; i<5; i++);
a++;
```

此处有分号

```c
/* while2.c -- watch your semicolons */
#include <stdio.h>
int main(void)
{
    int n = 0;

    while (n++ < 3);                /* line 7 */
    printf("n is %d\n", n);         /* line 8 */
    printf("That's all this program does.\n");

    return 0;
}
```

在不该打分号的地方应注意。
如应有分号，应另起一行引起注意。

```
n is 4
That's all this program does.
```

3. 关系运算符

关系和相等运算符

- 格式 <表达式1> <关系操作符> <表达式2>
- 关系表达式的值
  - 真：1；假：0

运算符	测试的关系
<	第一个操作数小于第二个操作数
>	第一个操作数大于第二个操作数
<=	第一个操作数小于或等于第二个操作数
>=	第一个操作数大于或等于第二个操作数
==	第一个操作数等于第二个操作数
!=	第一个操作数不等于第二个操作数

关系和相等运算符

- 注意
  - 浮点数存在舍入误差，因此对浮点数应慎用相等运算符
    - 因为2^n个位不为0，十进制小数转换为二进制无法用有限数字表示

```
( 1.1f - 1.f - .1f ) == 0     // 1.1000000238
```

```
fabs( 1.1f / 9 * 3 * 3 == 1.1f ) < 1e-6
```

$$1.09999990463256835 9375 = 1\times2^{0} + 1\times2^{-4} + 1\times2^{-5} + 1\times2^{-8} + 1\times2^{-9} + 1\times2^{-12} + 1\times2^{-13} + 1\times2^{-16} + 1\times2^{-17} + 1\times2^{-20} + 1\times2^{-21}$$

$$1.10000002384185791015625 = 1\times2^{0} + 1\times2^{-4} + 1\times2^{-5} + 1\times2^{-8} + 1\times2^{-9} + 1\times2^{-12} + 1\times2^{-13} + 1\times2^{-16} + 1\times2^{-17} + 1\times2^{-20} + 1\times2^{-21} + 1\times2^{-23}$$

$$0.09999999403953552246093 75, 0.10000000149011611938476 5625$$

```c
// cmpflt.c -- floating-point comparisons
#include <math.h>
#include <stdio.h>
int main(void)
{
    const double ANSWER = 3.14159;
    double response;
    printf("What is the value of pi?\n");
    scanf("%lf", &response);
    while (fabs(response - ANSWER) > 0.0001)
    {
        printf("Try again!\n");
        scanf("%lf", &response);
    }
    printf("Close enough!\n");

    return 0;
}
```

浮点数存在舍入误差，因此对浮点数应慎用相等运算符

```
What is the value of pi?
3.14↲
Try again!
3.14159265↲
Close enough!
That's all this program does.
```

```c
/* t_and_f.c -- true and false values in C */
#include <stdio.h>
int main(void)
{
    int true_val, false_val;

    true_val = (10 > 2);    // value of a true relationship
    false_val = (10 == 2);  // value of a false relationship
    printf("true = %d; false = %d \n", true_val, false_val);

    return 0;
}
```

```
true = 1; false = 0
```

逻辑运算符

- 格式 <表达式1> <逻辑操作符> <表达式2>
- 逻辑表达式的值
  - 真：1；假：0

var!=0和var
在逻辑上是等价的

运算符	描述	计算顺序
逻辑与 &&	如果两侧操作数的值都不等于0，则结果是1；否则结果是0。	第一个操作数等于0，则第二个操作数不会计算。
逻辑或 ||	如果两侧操作数的值都等于 0，则结果是0；否则结果是1。	第一个操作数不等于0，则第二个操作数不会计算。

```
if ( x < ++y && y++ < z )
    printf( "x is less than z\n" );
```

```c
// truth.c -- what values are true?
#include <stdio.h>
int main(void)
{
    int n = 3;

    while (n)
        printf("%2d is true\n", n--);
    printf("%2d is false\n", n);

    n = -3;
    while (n)
        printf("%2d is true\n", n++);
    printf("%2d is false\n", n);

    return 0;
}
```

在此 n!=0 和 n 的值是相同的。
如果判断数字是否为非0，建议写成 n!=0 ；如果判断逻辑值是否为真，建议写成n，避免歧义。

```
 3 is true
 2 is true
 1 is true
 0 is false
-3 is true
-2 is true
-1 is true
 0 is false
```

```c
// trouble.c -- misuse of =
// will cause infinite loop
#include <stdio.h>
int main(void)
{
    long num;
    long sum = 0L;
    int status;
    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    status = scanf("%ld", &num);
    while (status = 1)
    {
        sum = sum + num;
        printf("Please enter next integer (q to quit): ");
        status = scanf("%ld", &num);
    }
```

status = 1 的值恒为1。

```
    printf("Those integers sum to %ld.\n", sum);

    return 0;
}
```

```
Please enter an integer to be summed (q to quit): 3↲
Please enter next integer (q to quit): q↲
Please enter next integer (q to quit): Please enter next
integer (q to quit): Please enter next integer (q to quit):
Please enter next integer (q to quit): Please enter next
integer (q to quit):
```

逻辑表达式的真假值

- 逻辑表达式的值真为1，假为0
- 逻辑表达式的输入以“非零”为真，以0为假
- 调换书写顺序，避免因副作用为真（不是很推荐）

正写	反写	结果
`status = 1`	`1 = status`	反写时，少写了一个等号会提示编译错误。
`status == 1`	`1 == status`	正确。

```c
// trouble.c -- misuse of = will cause infinite loop
#include <stdio.h>
int main(void)
{
    long num, sum = 0L;
    int status;
    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    status = scanf("%ld", &num);
    while (1 == status)
    {
        sum = sum + num;
        printf("Please enter next integer (q to quit): ");
        status = scanf("%ld", &num);
    }
    printf("Those integers sum to %ld.\n", sum);
    retur
}
```

```
Please enter an integer to be summed (q to quit): 3↲
Please enter next integer (q to quit): q↲
Those integers sum to 3.
```

```c
// boolean.c -- using a _Bool variable
#include <stdio.h>
int main(void)
{
    long num;
    long sum = 0L;
    _Bool input_is_good;
    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    input_is_good = (scanf("%ld", &num) == 1);
    while (input_is_good)
    {
        sum = sum + num;
        printf("Please enter next integer (q to quit): ");
        input_is_good = (scanf("%ld", &num) == 1);
    }
    printf("Those integers sum to %ld.\n", sum);
    return 0;
}
```

```
Please enter an integer to be summed (q to
quit): 3↲
Please enter next integer (q to quit): 5↲
Please enter next integer (q to quit): q↲
Those integers sum to 8.
```

新的_Bool类型：只有1和0两种取值（不推荐新特性）

关系运算符的优先级

序	符号	运算类型	结合性	序	符号	运算类型	结合性
1	[ ] ( ) . -> 后缀++ 后缀--	表达式	从左到右	9	&	按位与	从左到右
				10	^	按位异或	从左到右
2	前缀++ 前缀-- sizeof * + - ~ !	一元	从右到左	11	|	按位或	从左到右
				12	&&	逻辑与	从左到右
3	强制类型转换	一元	从右到左	13	||	逻辑或	从左到右
4	* / %	乘法	从左到右	14	?:	条件表达式	从右到左
5	+ -	加法	从左到右	15	= *= /= %= += -= <<= >>= &= ^= |=	简单和复合	从右到左
6	<< >>	按位移动	从左到右				
7	< > <= >=	关系	从左到右				
8	== !=	相等	从左到右	16	,	顺序计算	从左到右

关系运算符的优先级

- 一元＜二元＜三元＜赋值＜逗号
- 一元：括号＜其它＜强制类型转换
- 二元：算术＜位移＜关系＜位＜逻辑，乘＜加
  - 乘除模＜加减；与＜或
  - 大小于＜相等
- 尊重常用：以常用方法不加括号为先。

4. 循环的灵活用法

不确定循环和计数循环

- 不确定循环 (indefinite loop)
  - 特点：不能预先知道循环了多少次
- 计数循环 (counting loop)
  - 特点：存在计数器，初始化，判断条件，更新计数
- for循环可以在一行内完成上述功能

```
for（初始状态；循环表达式；状态改变）
{
    循环体;
}
```

```c
// sweetie1.c -- a counting loop
#include <stdio.h>
int main(void)
{
    const int NUMBER = 22;
    int count = 1;                          // initialization

    while (count <= NUMBER)                 // test
    {
        printf("Be my Valentine!\n");       // action
        count++;                            // update count
    }

    return 0;
}
```

```
Be my Valentine!
Be my Valentine!
Be my Valentine!
（此处省略数行）
Be my Valentine!
```

```c
// sweetie2.c -- a counting loop using for
#include <stdio.h>
int main(void)
{
    const int NUMBER = 22;
    int count;

    for (count = 1; count <= NUMBER; count++)
        printf("Be my Valentine!\n");

    return 0;
}
```

仔细检查开始和结束的条件
注意是<=和<之间的差异。

```
Be my Valentine!
Be my Valentine!
Be my Valentine!
（此处省略数行）
Be my Valentine!
```

for循环的使用方法

- 语句之间可以互换，表达式之间可以互换
  - 但表达式不可以使用语句替换；语句可以使用表达式语句

方法	示例
减量运算符	`for (i = 5; i >= 0; --i)`
一次加减多步	`for (i = 5; i >= 0; i -= 2)`
用其它类型计数	`for (i = 'a'; i <= 'z'; ++i)`
其它循环条件	`for (i = 5; i * i * i >=1; i -= 2)`
算术或几何增加	`for (i = 0; i <= 10; i *= 1.2)`
用任何合法的表达式	`for (printf("Enter a number: "); scanf("%d",&num); printf("Enter again: "));`
任何一个表达式为空	`for (; i >= 0; i -= 2)` `for (;;)`

* 前、后自增的效率问题

扩展内容

- ++i 相当于下列代码（效率较高）

```
i = i + 1;
return i;
```

- i++ 相当于下列代码

```
j = i;
i = i + 1;
return j;
```

- 使用编译器的优化选项，会将该效率差别消除

```c
/* for_cube.c -- using a for loop to make a table of cubes */
#include <stdio.h>
int main(void)
{
    int num;

    printf("    n   n cubed\n");
    for (num = 1; num <= 6; num++)
        printf("%5d %5d\n", num, num*num*num);

    return 0;
}
```

```
    n   n cubed
    1       1
    2       8
    3      27
    4      64
    5     125
    6     216
```

逗号运算符

- 不便写入多个语句时，可用逗号分隔，形成一条语句
- 整个逗号表达式的值是右边的子表达式的值
- 逗号是一个序列点

```
x = 2.1, y = 1.3, z = 1.5;
x = ( y = 3, (z = ++y + 2) +5 );
```

```
x=11.000; y= 4.000; z= 6.000
```

- 注意不要把小数点错写成逗号

```
x = 2,1;
```

```c
// postage.c -- first-class postage rates
#include <stdio.h>
int main(void)
{
    const int FIRST_OZ = 46; // 2013 rate
    const int NEXT_OZ = 20;  // 2013 rate
    int ounces, cost;

    printf(" ounces  cost\n");
    for (ounces=1, cost=FIRST_OZ; ounces <= 16; ounces++,
         cost += NEXT_OZ)
        printf("%5d   $%4.2f\n", ounces, cost/100.0);

    return 0;
}
```

```
 ounces  cost
    1   $0.46
    2   $0.66
    3   $0.86
（此处省略数行）
   16   $3.46
```

序列求和

- 古希腊哲学家Zeno（芝诺）说：一支箭从弓箭手手里射出去，理论上这只箭永远到达不了靶子上
  - 一开始这只箭飞到了弓箭手与靶子的中间，
  - 然后它飞到了中间与靶子的位置（1/2），
  - 接着它飞到中间与靶子的中间位置（1/4），
  - 接着是（1/8），
  - 以此类推，它永远是在某个点与靶子中间的位置，它永远到不了靶子上

```c
/* zeno.c -- series sum
#include <stdio.h>
int main(void)
{
    int t_ct;       // t
    double time, power_of_2;
    int limit;
    printf("Enter the number of terms you want: ");
    scanf("%d", &limit);
    for (time=0, power_of_2=1, t_ct=1; t_ct <= limit;
                                t_ct++, power_of_2 *= 2.0)
    {
        time += 1.0/power_of_2;
        printf("time = %f when terms = %d.\n", time, t_ct);
    }
    return 0;
}
```

```
Enter the number of terms you want: 7↲
time = 1.000000 when terms = 1.
time = 1.500000 when terms = 2.
time = 1.750000 when terms = 3.
（此处省略数行）
time = 1.984375 when terms = 7.
```

```c
/* do_while.c -- exit condition loop */
#include <stdio.h>
int main(void)
{
    const int secret_code = 13;
    int code_entered;
    do
    {
        printf("To enter the triskaidekaphobia therapy club,\n");
        printf("please enter the secret code number: ");
        scanf("%d", &code_entered);
    } while (code_entered != secret_code);
    printf("Congratulations! You are cured!\n");
    return 0;
}
```

```
To enter the triskaidekaphobia therapy club,
please enter the secret code number: 32↲
To enter the triskaidekaphobia therapy club,
please enter the secret code number: 23↲
To enter the triskaidekaphobia therapy club,
please enter the secret code number: 13↲
Congratulations! You are cured!
```

```c
/* entry.c -- entry condition loop */
#include <stdio.h>
int main(void)
{
    const int secret_code = 13;
    int code_entered;
    printf("To enter the triskaidekaphobia therapy club,\n");
    printf("please enter the secret code number: ");
    scanf("%d", &code_entered);
    while (code_entered != secret_code)
    {
        printf("To enter the triskaidekaphobia therapy club,\n");
        printf("please enter the secret code number: ");
        scanf("%d", &code_entered);
    }
    printf("Congratulations! You are cured!\n");
    return 0;
}
```

5. 循环的中途退出

循环的中止

- 循环运行到break语句，跳出语句体。

```
1 始状态;]
while 2 条件表达式>)
{
   3 循环语句]
     if 4 退出条件>)
     5  break;
     [循环语句]
}
6 环体之外的语句]
```

```
for (1 始状态]; 2 件表达式]; [循环语句])
{
    3 循环语句（可包含状态的改变）]
     if 4 退出条件>)
     5  break;
     [循环语句（含状态的改变）]

}
6 环体之外的语句]
```

循环的继续

- continue语句中断本层循环，跳到判定条件
  - 如果为真，则进入循环；如果为假，则跳出循环

```
[初始状态;]
while (<条件表达式>)
{
    [循环语句]
    if (<退出条件>)
        continue;
    [循环语句]
}
[循环体之外的语句]
```

```
for ([初始状态]; <条件表达式>; [循环语句])
{
    [循环语句]
    if (<退出条件>)
        continue;
    [循环语句（含状态的改变）]
}
[循环体之外的语句]
```

嵌套循环（nest loop）

- 循环内部可以嵌套循环
  - 语句之间可以互换，表达式之间可以互换
  - 表达式不可以使用语句替换
  - 语句可以使用表达式语句表示
- 循环中的中止和继续语句只能退出当前循环

```c
/* rows1.c -- uses nested loops */
#include <stdio.h>
#define ROWS  6
#define CHARS 10
int main(void)
{
    int row;
    char ch;

    for (row = 0; row < ROWS; row++)
    {
        for (ch = 'A'; ch < ('A' + CHARS); ch++)
            printf("%c", ch);
        printf("\n");
    }

    return 0;
}
```

```
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
```

```c
// rows2.c -- using dependent nested loops
#include <stdio.h>
int main(void)
{
    const int ROWS = 6;
    const int CHARS = 6;
    int row;
    char ch;

    for (row = 0; row < ROWS; row++)
    {
        for (ch = ('A' + row); ch < ('A' + CHARS); ch++)
            printf("%c", ch);
        printf("\n");
    }

    return 0;
}
```

```
ABCDEF
BCDEF
CDEF
DEF
EF
F
```

内外层的循环条件之间不必相互独立

6. 循环的常见错误

循环前未设置初始状态

- 每次进入循环前应设置初始状态，否则结果不可控
- 循环后设置初始状态是不好的编程习惯

```
int sum = 0;

for (int i = 0; i < 10; i++)
{
    for (int j = 0; j < i; i++)
    {
        sum += arr[i];
    }
    sum = 0;
}
```

循环后设置初始状态是不好的编程习惯

```
for (int i = 0; i < 10; i++)
{
    int sum = 0;

    for (int j = 0; j < i; i++)
    {
        sum += arr[i];
    }
}
```

尽量在最小适用范围声明变量并初始化

循环使用的场合

- 流程大致相同，代码位置相邻
- 仅有次序区别，或没有区别

```
switch (month)
{
    case 1: total = 0; break;
    case 2: total = 31; break;
    case 3: total = 31 + 28; break;
    ...
    case 12: total = ...; break;
}
```

流程大致相同，仅有次序区别

```
int month_days[] = { 31,28,31,30,31,
    30,31,31,30,31,30,31 };
int total = 0;

for (int i = 0; i < month - 1; i++)
{
    total += month_days[i];
}
```

循环的终止条件错误常引发答案或者运行错误

循环的起止

- 循环的起止条件的书写应顺应变量的物理含义
  - 变量名和变量的使用应具备物理意义，否则误导读者

```
int month_days[] = { 31,28,31,30,31,
    30,31,31,30,31,30,31 };
int total = 0;

for (int i = 1; i < month; i++)
{
    total += month_days[i - 1];
}
```

变量i是数组下标，故意以1开始，下标使用i-1不直观

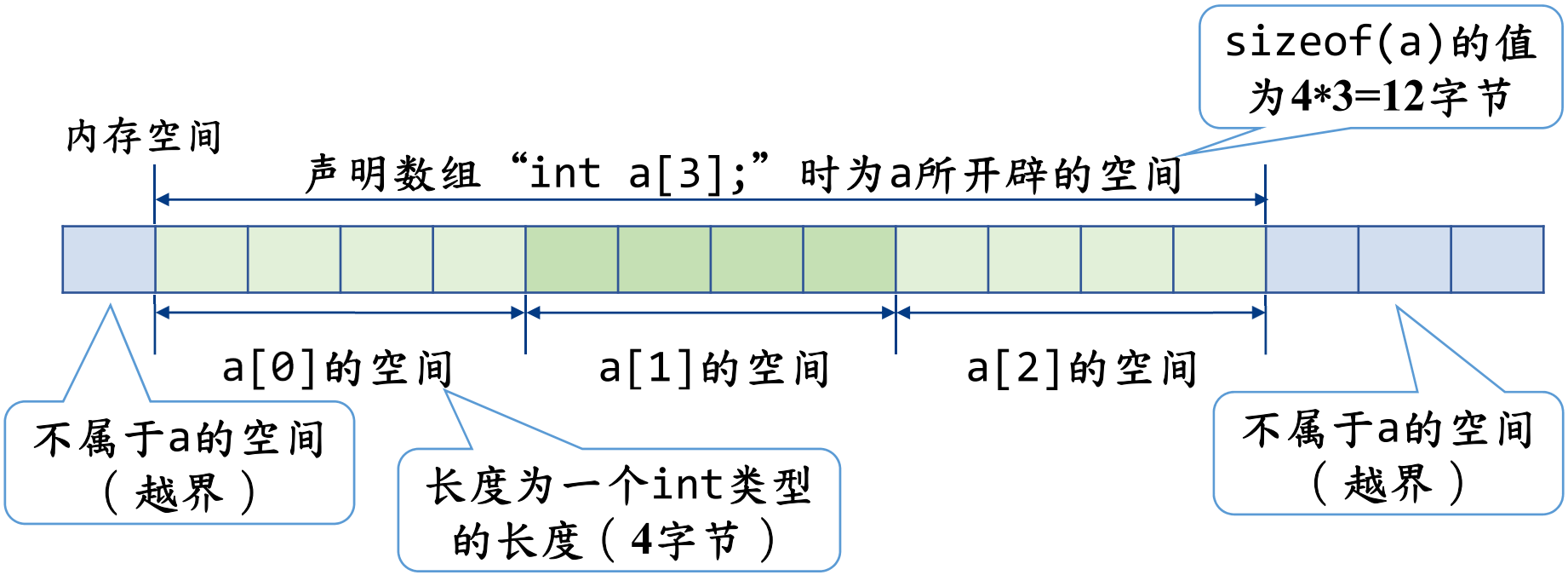
```
int month_days[] = { 31,28,31,30,31,
    30,31,31,30,31,30,31 };
int total = 0;

for (int i = 0; i < month - 1; i++)
{
    total += month_days[i];
}
```

循环变量为数组下标的，一般从0开始，循环条件使用小于号而不是小于等于号。

7. 数组初步

数组

- 数组的声明 <数据类型> <数组名字>[数组长度]
- 数组的引用 <数组名字>[下标]



```c
// scores_in.c -- uses loops for array processing
#include <stdio.h>
#define SIZE 10
#define PAR 72

int main(void)
{
        int index, score[SIZE];
        int sum = 0;
        float average;

        printf("Enter %d golf scores:\n", SIZE);
        for (index = 0; index < SIZE; index++)
                scanf("%d", &score[index]);  // read ten scores
        printf("The scores read in are as follows:\n");
        for (index = 0; index < SIZE; index++)
                printf("%5d", score[index]); // verify input
```

```
        printf("\n");
        for (index = 0; index < SIZE; index++)
                sum += score[index];          // add them up
        average = (float)sum / SIZE;     // time-honored method
        printf("Sum of scores = %d, average = %.2f\n", sum,
average);
        printf("That's a handicap of %.0f.\n", average - PAR);
        return 0;
}
```

输入11个数，但是有效的只有10个

```
Enter 10 golf scores:
23 10 99 95 64 67 13↲
78 85 56 82↲
The scores read in are as follows:
   23   10   99   95   64   67   13   78   85   56
Sum of scores = 590, average = 59.00
That's a handicap of -13.
```

循环计算数值的整数次幂

- 应设计一个计算次幂的函数
- 循环以输入是否合法为条件
  - 有些人以输入-1或0为退出条件，错的
  - 正常输入应该返回正确的结果
- 有一定的提示信息

```c
// power.c -- raises numbers to integer powers
#include <stdio.h>
double power(double n, int p); // ANSI prototype
int main(void)
{
    double x, xpow;
    int exp;
    printf("Enter a number and the positive integer power");
    printf(" to which\nthe number will be raised. Enter q");
    printf(" to quit.\n");
    while (scanf("%lf%d", &x, &exp) == 2)
    {
        xpow = power(x,exp);    // function call
        printf("%.3g to the power %d is %.5g\n", x, exp, xpow);
        printf("Enter next pair of numbers or q to quit.\n");
    }
    printf("Hope you enjoyed this power trip -- bye!\n");
    return 0;
}
```

函数要先声明，再定义，后使用

有经验的程序员先判断scanf()的返回值再处理程序

```
double power(double n, int p)  // function definition
{
    double pow = 1;
    int i;
    for (i = 1; i <= p; ++i)
        pow *= n;
    return pow;                // return the value of pow
}
```

```
Enter a number and the positive integer power to which
the number will be raised. Enter q to quit.
3 5↲
3 to the power 5 is 243
Enter next pair of numbers or q to quit.
6↲
3↲
6 to the power 3 is 216
Enter next pair of numbers or q to quit.
q↲
Hope you enjoyed this power trip -- bye!
```

C程序设计

T06

谢谢

厦门大学信息学院软件工程系

黄炜 副教授