C/C++ 语言

循环

张晓平

武汉大学数学与统计学院

Table of contents

示例程序

示例程序

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

示例程序

```
Enter an integer to be summed(q to quit): 22
Enter next integer(q to quit): 33
Enter next integer(q to quit): 44
Enter next integer(q to quit): q
Those integers sum to 99.
```

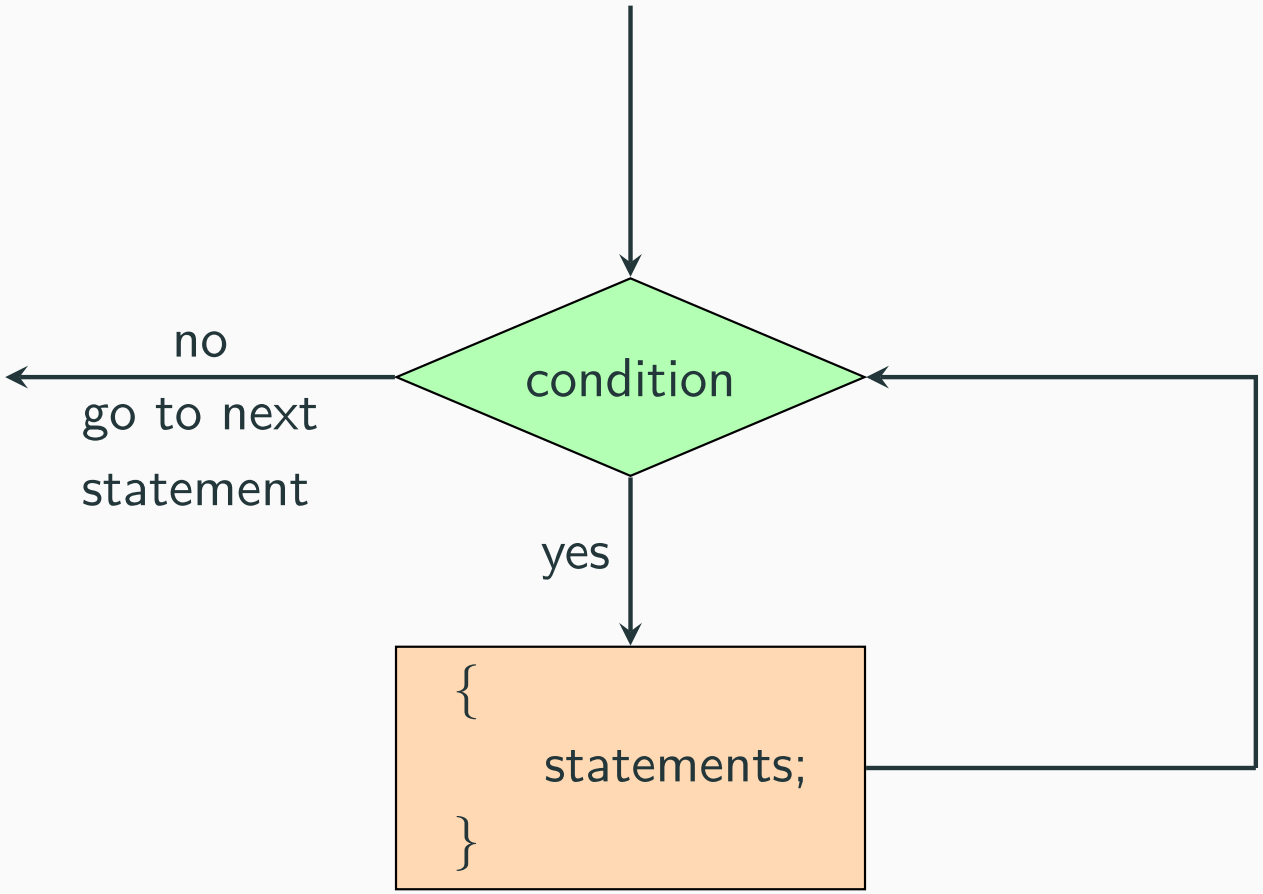
while语句

while语句

```
while (condition)
  statement
```

```
while (condition)
{
  statements
}
```

while语句



while语句：终止while循环

构造一个while循环时，必须能改变判断表达式的值，并最终使其为假，否则循环永远不会终止。

while语句：终止while循环

```
index = 1;
while (index < 5)
{
  printf("Good morning!\n");
}
```

while语句：终止while循环

```
index = 1;
while (index < 5)
{
  printf("Good morning!\n");
}
```

这一段代码无法终止循环，因为在循环中不能改变 index 的值。

while语句：终止while循环

```
index = 1;
while (--index < 5)
{
  printf("Good morning!\n");
}
```

while语句：终止while循环

```
index = 1;
while (--index < 5)
{
  printf("Good morning!\n");
}
```

虽然改变了 index 的值，但却朝着错误的方向，故仍无法退出循环。

while语句：终止while循环

```
index = 1;
while (++index < 5)
{
  printf("Good morning!\n");
}
```

while语句：终止while循环

```
index = 1;
while (++index < 5)
{
  printf("Good morning!\n");
}
```

这段代码可以正常退出循环。

while语句：何时终止循环

只有在计算判断条件的值时才能决定是否终止循环。

while语句：何时终止循环

```
1  // when.c:
2  #include <stdio.h>
3  int main(void)
4  {
5    int n = 5;
6    while (n < 7) {
7      printf("n = %d\n", n);
8      n++;
9      printf("Now n = %d\n", n);
10   }
11   return 0;
12 }
```

```
n = 5
Now n = 6
n = 6
Now n = 7
```

while语句：while：入口条件循环

while循环是使用入口条件的有条件循环。

while语句：何时终止循环

```
index = 10;
while (index++ < 5)
  printf("Have a fair day or better.\n");
```

while语句：何时终止循环

```
index = 10;
while (index++ < 5)
  printf("Have a fair day or better.\n");
```

把第一行改为

```
index = 3;
```

就可以执行这个循环了。

while语句：语法要点

在使用while时，请确定循环体的范围。缩进是为了帮助读者而不是计算机。

while语句：语法要点

```
1  // while1.c:
2  #include <stdio.h>
3  int main(void)
4  {
5    int n = 0;
6    while (n < 3)
7      printf("n = %d\n", n);
8      n++;
9      printf("That's all this"
10         " program does.\n");
11   return 0;
12 }
```

while语句：语法要点

```
1  // while1.c:
2  #include <stdio.h>
3  int main(void)
4  {
5    int n = 0;
6    while (n < 3)
7      printf("n = %d\n", n);
8      n++;
9      printf("That's all this"
10         " program does.\n");
11   return 0;
12 }
```

```
n = 0
n = 0
n = 0
n = 0
...
```

while语句：语法要点

while语句在语法上算作一条单独的语句，即使它使用了复合语句。该语句从while开始，到第一个分号结束；在使用了复合语句的情况下，到终结花括号结束。

while语句：语法要点

```
1  // while2.c:
2  #include <stdio.h>
3  int main(void)
4  {
5    int n = 0;
6    while (n++ < 3);
7      printf("n = %d\n", n);
8    printf("That's all this program does.\n");
9    return 0;
10 }
```

while语句：语法要点

```
1  // while2.c:
2  #include <stdio.h>
3  int main(void)
4  {
5    int n = 0;
6    while (n++ < 3);
7      printf("n = %d\n", n);
8    printf("That's all this program does.\n");
9    return 0;
10 }
```

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

while语句：语法要点

在 C 语言中，**单独的分号代表空语句 (null statement)**。

while语句：语法要点

有些时候，程序员会有意地使用带空语句的while语句。例如，假定你想要跳过输入直到第一个不为空或数字的字符，可以这样做。

while语句：语法要点

有些时候，程序员会有意地使用带空语句的while语句。例如，假定你想要跳过输入直到第一个不为空或数字的字符，可以这样做。

```
while(scanf("%d",&num)==1)
   ;
```

while语句：语法要点

有些时候，程序员会有意地使用带空语句的while语句。例如，假定你想要跳过输入直到第一个不为空或数字的字符，可以这样做。

```
while(scanf("%d",&num)==1)
   ;
```

请注意，为了清楚起见，请把分号单独置于while的下一行。

关系运算符和表达式

关系运算符和表达式

表 1: 关系运算符

运算符	含义
<	小于
<=	小于或等于
>	大于
>=	大于或等于
==	等于
!=	不等于

关系运算符和表达式

关系运算符用来构成`while`语句和其它 C 语句中使用的关系表达式，这些语句检查表达式是真还是假。

关系运算符和表达式

```
while(number < 6) {
  printf("Your number is too small.\n");
  scanf("%d", &number);
}
```

关系运算符和表达式

```
while(number < 6) {
  printf("Your number is too small.\n");
  scanf("%d", &number);
}
```

```
while(ch != '*') {
  count++;
  scanf("%c", &ch);
}
```

关系运算符和表达式

```
while(number < 6) {
  printf("Your number is too small.\n");
  scanf("%d", &number);
}
```

```
while(ch != '*') {
  count++;
  scanf("%c", &ch);
}
```

```
while(scanf("%f", &num) == 1)
  sum = sum + num;
```

关系运算符和表达式

- 不能使用关系运算符来比较字符串。

关系运算符和表达式

- 不能使用关系运算符来比较字符串。
- 关系运算符可用于浮点数。但要小心，在浮点数比较时只能使用 < 和 >，原因在于舍入误差可能造成两个逻辑上应该相等的数不相等。

  比如，虽然从数学上看

  $$3 * \frac{1}{3} == 1.0$$

  但若用 6 位小数表示 1/3，其乘积为 0.999 999。

关系运算符和表达式

- 不能使用关系运算符来比较字符串。
- 关系运算符可用于浮点数。但要小心，在浮点数比较时只能使用 < 和 >，原因在于舍入误差可能造成两个逻辑上应该相等的数不相等。

  比如，虽然从数学上看

  $$3 * \frac{1}{3} == 1.0$$

  但若用 6 位小数表示 1/3，其乘积为 0.999 999。
- 请使用头文件 `math.h` 中的 `fabs()` 来进行浮点数的判断，该函数返回一个浮点值的绝对值。

关系运算符和表达式

```c
// cmpflt.c:
#include <stdio.h>
#include <math.h>
int main(void)
{
  const double PI = 3.14159;
  double response;
  printf("What is the value of pi?\n");
  scanf("%lf", &response);
  while (fabs(response-PI)>0.0001) {
    printf("Try again!\n");
    scanf("%lf", &response);
  }
  printf("Close enough!\n");
  return 0;
}
```

关系运算符和表达式

```
What is the value of pi?
3.14
Try again!
3.1416
Close enough!
```

关系运算符和表达式：什么是真？

```c
// t_and_f.c
#include <stdio.h>
int main(void)
{
  int true_val, false_val;
  true_val  = (10 > 2);
  false_val = (10 == 2);
  printf("true = %d; false = %d\n",
     true_val, false_val);
  return 0;
}
```

关系运算符和表达式：什么是真？

```
// t_and_f.c
#include <stdio.h>
int main(void)
{
  int true_val, false_val;
  true_val  = (10 > 2);
  false_val = (10 == 2);
  printf("true = %d; false = %d\n",
     true_val, false_val);
  return 0;
}
```

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

关系运算符和表达式：什么是真？

对 C 来说，一个真表达式的值为 1，而一个假表达式的值为 0。

关系运算符和表达式：什么是真？

对 C 来说，一个真表达式的值为 1，而一个假表达式的值为 0。

```
while (1) {
   ...
}
```

死循环

关系运算符和表达式：还有什么是真？

问题

既然可以使用 `1` 或 `0` 来作为 `while` 语句的判断表达式，那么还可以使用其他数字吗？

关系运算符和表达式：还有什么是真？ i

```c
// truth.c
#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;
}
```

关系运算符和表达式：还有什么是真？

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

关系运算符和表达式：还有什么是真？

对 C/C++ 来说，所有非零值都被认为是真，只有0被认为是假。

关系运算符和表达式：还有什么是真？

对 C/C++ 来说，所有非零值都被认为是真，只有0被认为是假。

基于上述认识可知，如下两种方式等价：

```
while (n != 0) {
   ...
}
```

```
while (n) {
   ...
}
```

关系运算符和表达式：真值的问题

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

关系运算符和表达式：真值的问题

```
Enter an integer to be summed (q to quit):
1
Enter next integer (q to quit):
2
Enter next integer (q to quit):
3
Enter next integer (q to quit):
q
Enter next integer (q to quit):
Enter next integer (q to quit):
Enter next integer (q to quit):
...
```

关系运算符和表达式：真值的问题

该例改变了while的判断条件，用 status=1 代替了 status==1。而赋值表达式的值就是其左侧的值，故 status=1 的值为 1。实际上，

```
while (status = 1)
```

等价于

```
while (1)
```

进入死循环。

关系运算符和表达式：真值的问题

- 当你输入 `q` 后，根本没有机会进行更多的输入。
- 当 `scanf()` 未能读取指定形式的输入时，它就留下这个不相容的输入，以供下次进行读取。
- 当 `scanf()` 试着把 `q` 当做整数读取并失败时，它就把 `q` 留在那里。下次循环继续读取这个 `q`，`scanf()` 再次失败。
- 该例不但建立了一个无限循环，更建立了一个无限失败的循环。

关系运算符和表达式：真值的问题

- 不要在应该使用 == 的地方使用 =。
- 赋值运算符 = 把一个值赋给左边的变量，而关系运算符 == 检查左右两边的值是否相等，它并不改变左边变量的值。

```
i = 5    // 把 i 赋值为 5
i == 5   // 检查 i 的值是否为 5
```

- 在使用 == 时，若比较双方有一个是常量，可以把它放在左侧，以便于发现错误。

```
5 = i    // 语法错误
5 == i   // 检查 i 的值是否为 5
```

关系运算符和表达式：真值小结

- 关系运算符用于构成关系表达式，关系表达式为真时值为 1，为假时值为 0。
- 对于使用关系表达式作为判断条件的语句（while 和 if），可以使用任何表达式作为判断，非零值被认为是真，而零值被认为是假。

关系运算符和表达式：_Bool类型

- 表示真/假的变量被称为布尔变量，在 C 中一直由 int 类型来表示。
- C99 添加了 _Bool 类型，是布尔变量的类型名。
- 一个布尔变量只可以具有值 1 或 0。若把一个布尔变量赋为非零数值，它就被设置为 1。这说明 C 把任何非零值都认为是真。

关系运算符和表达式：_Bool类型

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

关系运算符和表达式：_Bool类型

```
input_is_good = (scanf("%ld", &num) == 1);
```

- 该代码将比较的结果赋给布尔变量。
- 把 == 表达式括起来的括号不是必需的，因为 == 的优先级比 = 要高，但这样写可使代码更容易阅读。
- 还请注意布尔变量的命名方式：

```
while (input_is_good)
```

关系运算符和表达式：_Bool类型

C99 还提供了一个头文件 stdbool.h。使用它可以用 bool 代替_Bool，并把 true 和 false 定义成值为 1 和 0 的常量。

关系运算符和表达式：_Bool类型

```c
// boolean1.c:
#include <stdio.h>
#include <stdbool.h>
int main(void)
{
  long num;
  long sum = 0L;
  bool input_is_good;
  printf("Enter an integer to be summed");
  printf(" (q to quit):\n");
  input_is_good = (scanf("%ld", &num) == true);
  while (input_is_good) {
    sum = sum + num;
    printf("Enter next integer (q to quit):\n");
    input_is_good = (scanf("%ld", &num) == true);
  }
  printf("Those integers sum to %ld.\n", sum);
  return 0;
}
```

关系运算符和表达式：关系运算符的优先级

关系运算符的优先级低于包括 + 和 - 在内的算术运算符，但要高于赋值运算符。如

```
x > y+2          ⟺   x > (y+2)

x = y > 2        ⟺   x = (y > 2)

x_bigger = x > y ⟺   x_bigger = (x > y)
```

关系运算符和表达式：关系运算符的优先级

表 2：关系运算符本身有两组不同的优先级

高优先级	< <= > >=
低优先级	== !=

结合规则：从左到右。

关系运算符和表达式：关系运算符的优先级

```
c != a == b
```

$\Longleftrightarrow$

```
(c != a) == b
```

关系运算符和表达式：运算符的优先级

运算符（从高到低）	结合性
`()`	从左到右
`- + ++ -- sizeof (type)`	从右到左
`* / %`	从左到右
`+ -`	从左到右
`< <= > >=`	从左到右
`== !=`	从左到右
`=`	从右到左

不确定循环与计数循环

不确定循环与计数循环

- 不确定循环：在表达式变为假之前你不能预知循环要执行多少次。
- 计数循环：执行预先确定的循环次数。

不确定循环与计数循环

```
// sweetie1.c:
#include <stdio.h>
int main(void)
{
  const int NUMBER = 2;
  int count = 1;
  while (count <= NUMBER) {
    printf("Hello world!\n");
    count++;
  }
  return 0;
}
```

不确定循环与计数循环：循环三要素

- 从哪里来（初始化计数器）
- 到哪里去（计数器与某个有限值做比较）
- 怎么走（每次执行循环，计数器要递增）

不确定循环与计数循环：循环三要素

```
while (count <= NUMBER){
   ...
   count++;
}
```

$\Longrightarrow$

```
while (count++ <= NUMBER){
   ...
}
```

for 循环

for 循环

```
for (initialization; condition; increment) {
  statements
}

for (initialization; condition; increment)
  statement
```

for 循环把初始化、测试与更新三个动作放在一起。

for 循环

```c
// sweetie2.c:
#include <stdio.h>
int main(void)
{
  const int NUMBER = 4;
  int count;

  for (count = 0; count < NUMBER; count++)
    printf("Hello world!\n");

  return 0;
}
```

for 循环

```
Hello world!
Hello world!
Hello world!
Hello world!
```

for 循环

`for` 后面的圆括号中包含由两个分号隔开的三个表达式。

1. 第一个表达式进行初始化，在 `for` 循环开始时执行一次。
2. 第二个表达式是判断条件，在每次执行循环前都要对它求值，值为假时，循环结束。
3. 第三个表达式用于更新，在每次循环结束时进行计算。

三个表达式中的每一个都是完整的，故任意一个表达式的副作用都在程序求下一个表达式的值前生效。

for 循环

```
#include <stdio.h>
int main(void)
{
  int num;
  printf("%3s %8s\n", "n", "n cubed");
  for (num = 1; num <= 4; num++)
    printf("%3d %8d\n", num, num*num*num);
  return 0;
}
```

for 循环

```
  n   n cubed
  1         1
  2         8
  3        27
  4        64
```

for 循环：利用 for 的灵活性

1、可以使用自减运算符来减小计数器。

```
// for_down.c:
#include <stdio.h>
int main(void)
{
  int secs;
  for (secs = 4; secs > 0; secs--)
    printf("%d seconds!\n", secs);
  printf("Ignition!\n");
  return 0;
}
```

for 循环：利用 for 的灵活性

```
4 seconds!
3 seconds!
2 seconds!
1 seconds!
Ignition!
```

for 循环：利用 for 的灵活性

2、若有需要，可让计数器一次加 2，加 10，等等。

```
// for_13s.c:
#include <stdio.h>
int main(void)
{
  int n;
  for (n = 2; n < 60; n = n+13)
    printf("%d\n", n);
  return 0;
}
```

for 循环：利用 for 的灵活性

```
2
15
28
41
54
```

for 循环：利用 for 的灵活性

3、可让字符代替数字进行计数。

```
// for_char.c:
#include <stdio.h>
int main(void)
{
  char ch;
  for (ch = 'a'; ch <= 'z'; ch++)
    printf("The ASCII value of %c is %d\n", ch,
    ch);
  return 0;
}
```

for 循环：利用 for 的灵活性

```
The ASCII value of a is 97
The ASCII value of b is 98
...
The ASCII value of y is 121
The ASCII value of z is 122
```

for 循环：利用 for 的灵活性

4、可判断迭代次数之外的条件。

```
// for_cube1.c:
#include <stdio.h>
int main(void)
{
  int num;
  printf("%3s %8s\n", "n", "n cubed");
  for (num = 1; num*num*num <= 64; num++)
    printf("%3d %8d\n", num, num*num*num);
  return 0;
}
```

for 循环：利用 for 的灵活性

```
n   n cubed
1         1
2         8
3        27
4        64
```

for 循环：利用 for 的灵活性

5、可让数量几何增加而不是算术增加；即不是每一次加一个固定的数，而是乘上一个固定的数。

```
// for_geo.c:
#include <stdio.h>
int main(void)
{
  double debt;
  for (debt = 100.0; debt <= 140; debt=debt*1.1)
    printf("Your debt is now $%.2f\n", debt);
  return 0;
}
```

for 循环：利用 for 的灵活性

```
Your debt is now $100.00
Your debt is now $110.00
Your debt is now $121.00
Your debt is now $133.10
```

for 循环：利用 for 的灵活性

6、第三个表达式中，可以使用任何合法表达式。

```
// for_wild.c:
#include <stdio.h>
int main(void)
{
  int x;
  int y = 55;
  for (x = 1; y <= 75; y = (++x * 5) + 50)
    printf("%10d %10d\n", x, y);
  return 0;
}
```

for 循环：利用 for 的灵活性

```
1       55
2       60
3       65
4       70
5       75
```

for 循环：利用 for 的灵活性

7、甚至可以让一个或多个表达式为空，但不要遗漏分号。只需确保在循环中包含了一些能使循环最终结束的语句。

for 循环：利用 for 的灵活性

```c
// for_none.c:
#include <stdio.h>
int main(void)
{
  int ans, n;
  ans = 2;
  for (n = 3; ans <= 25; )
    ans = ans * n;
  printf("n = %d; ans = %d.\n", n, ans);
  return 0;
}
```

for 循环：利用 for 的灵活性

```
n = 3; ans = 54.
```

for 循环：利用 for 的灵活性

第二个表达式为空会被认为是真，故下面的循环会永远执行：

```
for ( ; ; )
  printf("I want som action\n");
```

for 循环：利用 for 的灵活性

8、第一个表达式不必初始化一个变量，它也可以是某种类型的 printf 语句。请记住第一个表达式只在执行循环的其他部分之前被求值或执行一次。

for 循环：利用 for 的灵活性

```
// for_show.c:
#include <stdio.h>
int main(void)
{
  int num;
  for (printf("Keep entering numbers!\n");
       num!=6;  )
    scanf("%d", &num);
  printf("That's the one I want!\n");
  return 0;
}
```

for 循环：利用 for 的灵活性

```
Keep entering numbers!
2
6
That's the one I want!
```

for 循环：利用 for 的灵活性

9、循环中的动作可以改变循环表达式的参数。例如，假定有这样一个循环

```
for (n = 1; n < 10000; n = n + delta)
```

如果执行几次循环之后，程序觉得 delta 的值太小或太大，可以结合 if 语句改变 delta 的大小。

更多赋值运算符

更多赋值运算符

```
+=    -=    *=    /=    %=
```

更多赋值运算符

```
num += 20
num -= 20
num /= 20
num *= 20
num %= 20
```

⇔

```
num = num + 20
num = num - 20
num = num / 20
num = num * 20
num = num % 20
```

更多赋值运算符

```
num += 20
num -= 20
num /= 20
num *= 20
num %= 20
```

⇔

```
num = num + 20
num = num - 20
num = num / 20
num = num * 20
num = num % 20
```

```
x *= 3 * y + 12
```

⇔

```
x = x * (3 * y + 12)
```

更多赋值运算符

- 这些运算符具有与 = 同样低的优先级。
- 使用这些运算符可以让代码更为简洁，与长形式相比可能会产生效率更高的机器代码。特别是变量名很长时，使用它们就显得非常有必要。如

```
xxxxyyyyzzzz *= 3
```

```
xxxxyyyyzzzz = xxxxyyyyzzzz * 3
```

逗号运算符

逗号运算符

逗号运算符扩展了 `for` 循环的灵活性，它使你可以在一个 `for` 循环中使用多个初始化或更新表达式。

逗号运算符

例
编制程序，打印一类邮资费率。费用标准为：第 1 盎司为 37 美分，然后每增加 1 盎司费用增加 23 美分。

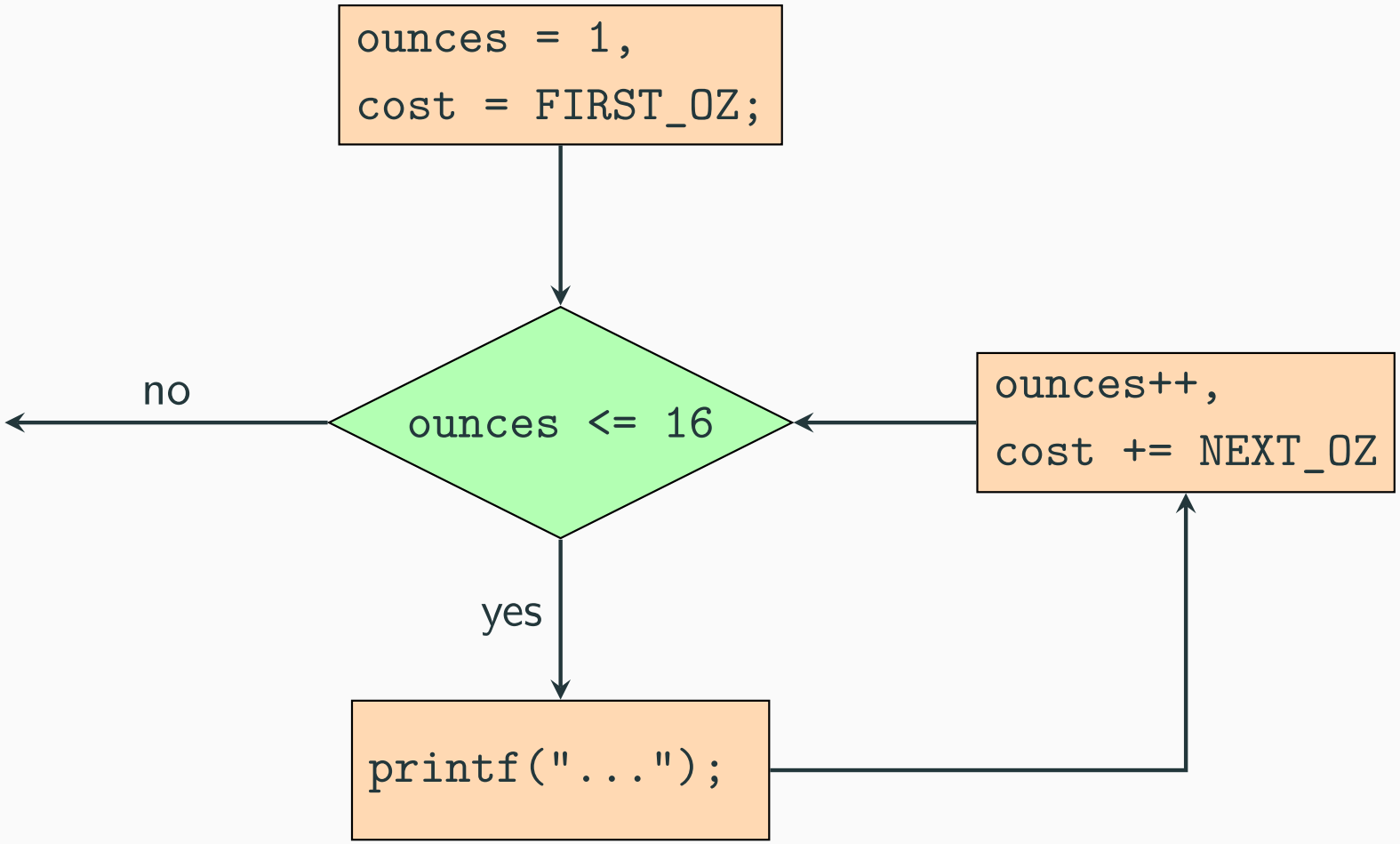
逗号运算符

```
1  // postage.c:
2  #include <stdio.h>
3  int main(void)
4  {
5    const int FIRST_OZ = 37, NEXT_OZ = 23;
6    int ounces, cost;
7    for (ounces = 1, cost = FIRST_OZ;
8         ounces <= 16;
9         ounces++, cost += NEXT_OZ)
10     printf("%2d $%4.2f\n", ounces, cost/100.0);
11   return 0;
12 }
```

逗号运算符

```
1 $0.37
2 $0.60
3 $0.83
  ...
```

逗号运算符


ounces = 1,
cost = FIRST_OZ;
ounces <= 16
no
yes
ounces++,
cost += NEXT_OZ
printf("...");

逗号运算符

- 逗号运算符并不只限于在 `for` 循环中使用，但在 `for` 循环中最常使用。

逗号运算符

- 逗号运算符保证被它分开的表达式按从左到右的次序进行计算。也就是说，逗号是个顺序点，逗号左边的副作用会在程序运行到逗号右边之前生效。

```
ounces++, cost = ounces * FIRST_OZ
```

逗号运算符

- 整个逗号表达式的值是右边成员的值。

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

的效果是

```
(1) y = 3;
(2) y = y + 1 = 4;
(3) z = (y + 2) = (4 + 2) = 6;
(4) x = z + 5 = 6 + 5 = 11;
```

逗号运算符

```
houseprice = 249,500;
```

等效于

```
houseprice = 249;
500;
```

逗号运算符

```
houseprice = 249,500;
```

等效于

```
houseprice = 249;
500;
```

C 把它解释为一个逗号表达式，houseprice = 249 为左子表达式，而 500 为右子表达式。因此整个逗号表达式的值为右子表达式的值 500，而左子表达式将变量 houseprice 赋值为 249。

逗号运算符

```
houseprice = (249,500);
```

逗号运算符

```
houseprice = (249,500);
```

把右子表达式的值 500 赋给变量 houseprice。

逗号运算符

逗号也用作分隔符。

```
int m, n;
printf("%d %d\n", m, n);
```

逗号运算符

逗号也用作分隔符。

```
int m, n;
printf("%d %d\n", m, n);
```

这里，逗号都是分隔符，而不是逗号运算符。

逗号运算符

例

计算

$$S = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots$$

逗号运算符

```
// zeno.c:
#include <stdio.h>
int main(void)
{
  int count, limit;
  double sum, x;
  printf("Enter the number of terms: ");
  scanf("%d", &limit);
  for (sum = 0., x = 1, count = 1;
       count <= limit;
       count++, x *= 2.) {
    sum += 1.0/x;
    printf("sum = %f when terms = %d.\n",
       sum, count);
  }
  return 0;
}
```

逗号运算符

```
Enter the number of terms you want: 10
sum = 1.000000 when terms = 1.
sum = 1.500000 when terms = 2.
sum = 1.750000 when terms = 3.
sum = 1.875000 when terms = 4.
sum = 1.937500 when terms = 5.
sum = 1.968750 when terms = 6.
sum = 1.984375 when terms = 7.
sum = 1.992188 when terms = 8.
sum = 1.996094 when terms = 9.
sum = 1.998047 when terms = 10.
```

退出条件循环 (do while)

退出条件循环 (do while)

- while 循环和 for 循环都是入口条件循环，在每次执行循环之前先检查判断条件，这样循环中的语句可能一次也不执行。
- do while 循环为退出条件循环，判断条件在执行循环之后进行检查，这可保证循环体中的语句至少被执行一次。

退出条件循环 (do while)

```c
// do_while.c:
#include <stdio.h>
int main(void)
{
  const int SECRET_CODE = 13;
  int code_entered;
  do {
    printf("To withdraw money from ATM. \n");
    printf("Enter the secret code number: ");
    scanf("%d", &code_entered);
  } while (code_entered != SECRET_CODE);
  printf("Congratulations! You are permitted!\n"
  );
  return 0;
}
```

退出条件循环 (do while)

```
To withdraw money from ATM.
Please enter the secret code number: 11
To withdraw money from ATM.
Please enter the secret code number: 12
To withdraw money from ATM.
Please enter the secret code number: 13
Congratulations! You are permitted!
```

退出条件循环 (do while) i

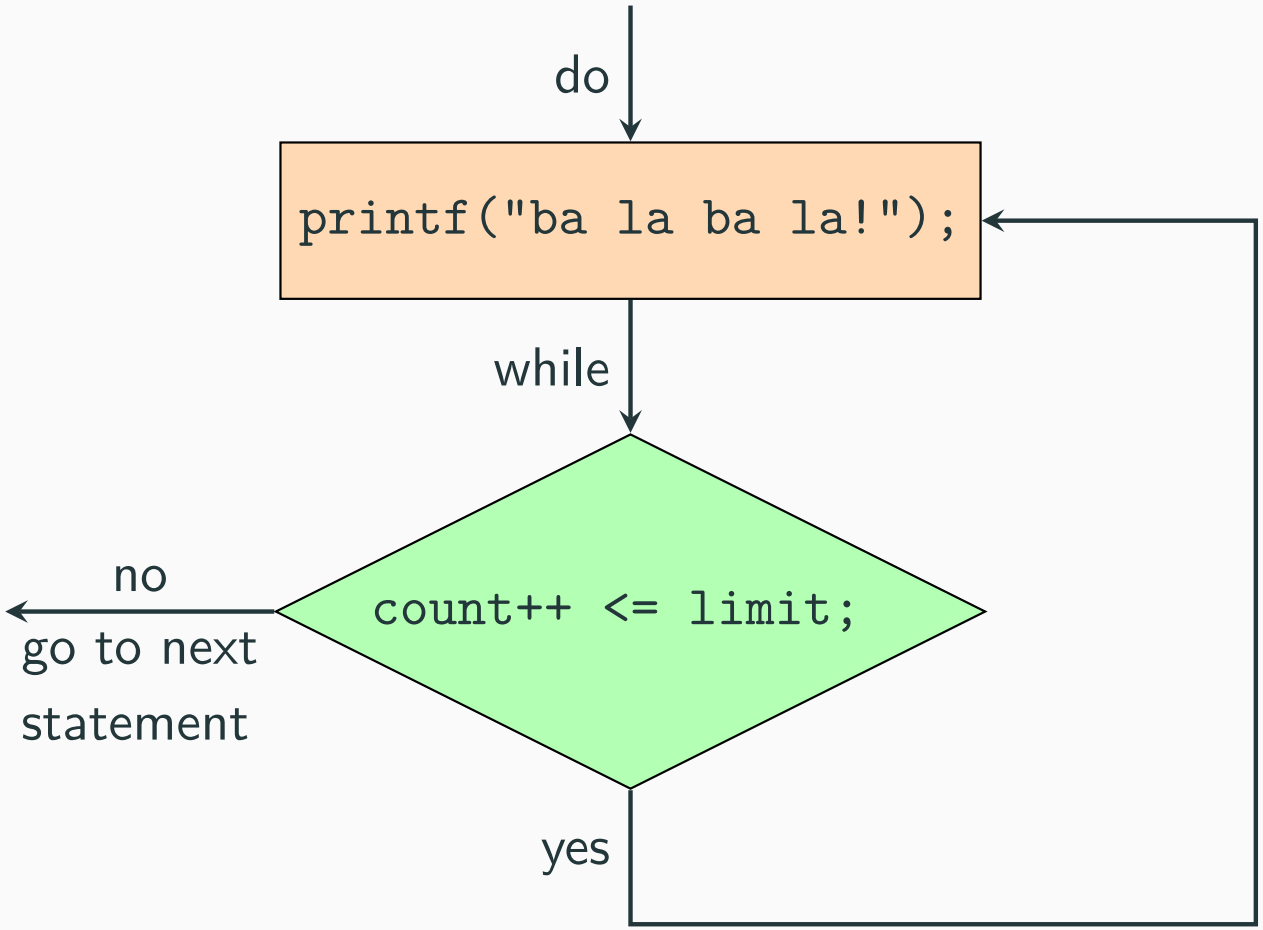
若用 while 循环改写这段程序，代码会长一些。

```
// entry.c:
#include <stdio.h>
int main(void)
{
  const int SECRET_CODE = 13;
  int code_entered;
  printf("To withdraw money from ATM. \n");
  printf("Enter the secret code number: ");
  scanf("%d", &code_entered);
  while (code_entered != SECRET_CODE) {
    printf("To withdraw money from ATM. \n");
    printf("Enter the secret code number: ");
    scanf("%d", &code_entered);
```

退出条件循环 (do while) ii

```
  }
  printf("Congratulations! You are permitted!\n"
  );
  return 0;
}
```

退出条件循环 (do while)



退出条件循环 (do while)

```
do {
        statements
} while (condition);

do
        statement
while (condition);
```

退出条件循环 (do while)

- 请注意do while本身是一条语句，它需要一个结束的分号。
- 应该把do while仅用于那些至少需要执行一次循环的情况。

选择哪种循环

选择哪种循环

先确定是入口条件循环还是退出条件循环。通常采用入口条件循环，原因在于

- 一般原则：在循环之前判断要比之后好；
- 在循环开始时进行判断，程序的可读性更强；
- 很多应用中，若一开始就不满足判断条件，那么跳过整个循环是非常重要的。

选择哪种循环

要使 `for` 循环更像 `while` 循环，可以去掉第一个和第三个表达式：

```
for (; condition; )
```

与

```
while (condition)
```

等效。

选择哪种循环

要使 while 循环更像 for 循环，可以在前面使用初始化语句并包含更新语句：

```
initialization;
while (condition) {
  body;
  update;
}
```

与

```
for (initialization; condition; update){
  body;
}
```

等效。

选择哪种循环

- 在循环中涉及到初始化和更新变量时，使用 `for` 循环较为适当，而在其他条件下使用 `while` 循环更好一些。`while` 循环对以下条件比较自然：

```
while (scanf("%ld", &num) == 1){
   ...
}
```

- 对那些涉及到用索引计数的循环，使用 `for` 循环是一个更自然的选择。

```
for (count = 1; count <= 100; count++){
   ...
}
```

嵌套循环 (nested loop)

嵌套循环 (nested loop)

嵌套循环是指在另一个循环内的循环。比如打印矩阵就需要用到嵌套循环。

嵌套循环 (nested loop)

```
// rows1.c:
#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;
}
```

嵌套循环 (nested loop)

```
A B C D E F G H I J
A B C D E F G H I J
A B C D E F G H I J
A B C D E F G H I J
A B C D E F G H I J
A B C D E F G H I J
```

嵌套循环 (nested loop)

```
// row2.c:
#include <stdio.h>
#define ROWS 6
#define CHARS 6
int main(void)
{
  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;
}
```

嵌套循环 (nested loop)

```
A B C D E F
B C D E F
C D E F
D E F
E F
F
```

数组

数组

数组是线性存储的一系列相同类型的值。整个数组有一个单一的名字，每个元素使用一个整数索引来进行访问。

数组：声明

```
float debts[20];
```

- 声明 debts 是一个具有 20 个元素的数组，每个元素都是一个类型为 float 的值。
- 该数组的第一个元素为 debts[0]，第二个元素为 debts[1]，...。
- 数组元素的编号从零开始。
- 每个元素都可以被赋予一个 float 类型的值。

```
debts[3] = 2.;
debts[7] = 1.2e+10;
```

数组

- 可以像使用相同类型的变量那样使用一个数组元素。例如，你可以把一个值读入一个特定的元素：

```
scanf("%d", &debts[4]); // 为第 5 个元素读入一个值
```

数组

- 易犯的错误：处于执行速度的考虑，C 并不检查你是否使用了正确的下标。如

```
debts[20] = 11.0;   // 没有这个数组元素
debts[31] = 22.22; // 没有这个数组元素
```

但编译器不会报错。当程序运行时，这些语句把数据放在可能由其他数据使用的位置上，因而可能破坏程序的结果甚至是程序崩溃。

数组

- 数组可以是任意数据类型。

```
int num[10];   // 一个存放 10 个整数的数组
char ch[20];   // 一个存放 20 个字符的数组
double a[40]; // 一个存放 40 个 double 值的数组
```

特别地，字符串就是一个字符数组。

数组

- 用于标识数组元素的数字称为下标 (subscript)、索引 (index) 或偏移量 (offset)。
- 下标必须是整数，它从 0 开始。
- 数组中的元素在内存中是顺序存储的。

在 for 循环中使用数组

例

读入你 5 天运动的步数，然后进行处理，最后报告步数的总和、平均值和差点（平均值与标志值之间的差）。

数组 i

```c
// scores.c:
#include <stdio.h>
#define DAYS 5
#define PAR  10000
int main(void)
{
  int index, steps[DAYS];
  int sum = 0;
  float average;
  printf("Enter steps of %d days:\n", DAYS);
  for (index = 0; index < DAYS; index++)
    scanf("%d", &steps[index]);
  printf("The steps read in are as follows:\n");
  for (index = 0; index < DAYS; index++)
```

```
    printf("%7d", steps[index]);
  printf("\n");
  for (index = 0; index < DAYS; index++)
    sum += steps[index];
  average = (float) sum / DAYS;
  printf("Sum of steps = %d, average = %.2f\n",
  sum, average);
  printf("That's a handicap of %.0f.\n", average
  -PAR);
  return 0;
}
```

数组

```
Enter steps of 5 days:
9999 9995 11000 12012 11145 95667
The steps read in are as follows:
   9999   9995  11000  12012  11145
Sum of steps = 54151, average = 10830.20
That's a handicap of 830.
```

使用函数返回值的循环

使用函数返回值的循环

例

编制一个函数，计算一个数的整数次幂，如 n^p。

使用函数返回值的循环

例

编制一个函数，计算一个数的整数次幂，如 n^p。

请大家记住，头文件 `math.h` 中提供了一个名为 `pow` 的幂函数，允许计算浮点数次幂。

pow 函数的使用

```c
// pow.c:
#include <stdio.h>
#include <math.h>
int main(void)
{
  printf("   2^3    = %f\n", pow(2.0,3.0));
  printf(" sqrt 2   = %f\n", pow(2.0,0.5));
  printf(" 3^(1/4) = %f\n", pow(3.0,0.25));
  return 0;
}
```

pow 函数的使用

```
  2^3    = 8.000000
sqrt 2   = 1.414214
3^(1/4)  = 1.316074
```

使用函数返回值的循环

```
1 double power(double n, int p)
2 {
3   double pow = 1;
4   int i;
5   for (i = 1; i <= p; i++)
6     pow *= n;
7   return pow;
8 }
```

使用函数返回值的循环

写一个具有返回值的函数需要做以下事情：

- 定义函数时，说明它的返回值类型；
- 使用关键字 `return` 指示要返回的值。

使用函数返回值的循环

- 要声明函数类型，可以在函数名之前写出类型，就像声明一个变量时那样；
- 关键字 `return` 使函数把它后面的值返回给调用函数。可以返回一个值，也可以返回一个表达式。

```
return 2 * x + b;
```

使用函数返回值的循环

在调用函数中，

- 可以把一个返回值赋给另一个变量。

```
b = power(1.2, 3);
```

- 可以把一个返回值作为一个表达式中的值。

```
b = 2.0 + power(1.2, 3);
```

- 也可以把一个返回值作为另一个函数的参数。

```
printf("%f", power(1.2, 3));
```

power 函数的测试 i

```
// power.c:
#include <stdio.h>
double power(double n, int p);
int main(void)
{
  double x, xpow;
  int exp;
  printf("Enter a number and the positive
  integer power\n");
  printf("Enter q to quit.\n");
  while (scanf("%lf%d", &x, &exp) == 2) {
    xpow = power(x, exp);
    printf("%.3g to the power %d is %.5g\n",
           x, exp, xpow);
```

power 函数的测试 ii

```
    printf("Enter next pair of numbers or q to
    quit.\n");
  }
  printf("Hope you enjoyed this power trip --
  bye!\n");
  return 0;
}
double power(double n, int p)
{
  double pow = 1;
  int i;
  for (i = 1; i <= p; i++)
    pow *= n;
  return pow;
}
```

power 函数的测试 iii

power 函数的测试

```
Enter a number and the positive integer power
to which
the number will be raised. Enter qto quit.
1.2 12
1.2 to the power 12 is 8.9161
Enter next pair of numbers or q to quit.
2
16
2 to the power 16 is 65536
Enter next pair of numbers or q to quit.
q
```