It makes C more powerful than maths.

C语言程序设计

C Programming

5

运算符、表达式和语句

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 炜

(特色化示范性软件学院) 博士,副教授

School of Informatics Wei Huang

内容要点

- 利用循环解决问题
- 运算符
  - 赋值、四则运算（加减乘除模）、复合（四则与赋值结合）
  - 强制类型转换、尺寸、增减量
  - 优先级
- 表达式
  - 序列点
- 语句

目录

```c
/* shoes1.c -- converts a shoe size to inches */
#include <stdio.h>
#define ADJUST 7.31      // one kind of symbolic constant
int main(void)
{
    const double SCALE = 0.333; // another kind of symbolic constant
    double shoe, foot;

    shoe = 9.0;
    foot = SCALE * shoe + ADJUST;
    printf("Shoe size (men's)    foot length\n");
    printf("%10.1f %15.2f inches\n", shoe, foot);

    return 0;
}
```

```
Shoe size (men's)    foot length
       9.0              10.31 inches
```

实例

- 题目
  - 输出3—18号的男鞋足长
- 样例输入：无
- 样例输出

```
Shoe size (men's)    foot length
        3.0           8.31 inches
        4.0           8.64 inches
（此处忽略一部分）
       18.0          13.30 inches
If the shoe fits, wear it.
```

- 最直接的想法
  - 尺寸从3开始，若小于等于18，依次完成
    - 根据号码计算足长；号码增加1步；回到循环

```c
/* shoes2.c -- calculates foot lengths for several sizes */
#include <stdio.h>
#define ADJUST 7.31      // one kind of symbolic constant
int main(void)
{
    const double SCALE = 0.333; // another kind of symbolic constant
    double shoe, foot;
    printf("Shoe size (men's)    foot length\n");
    shoe = 3.0;
    while (shoe < 18.5)
    {
        foot = SCALE * shoe + ADJUST;
        printf("%10.1f %15.2f inches\n", shoe, foot);
        shoe = shoe + 1.0;
    }
    printf("If the shoe fi
    return 0;
}
```

有经验的程序员不会在此写18.5这样令人不解的数

指定宽度可以使得输出更美观

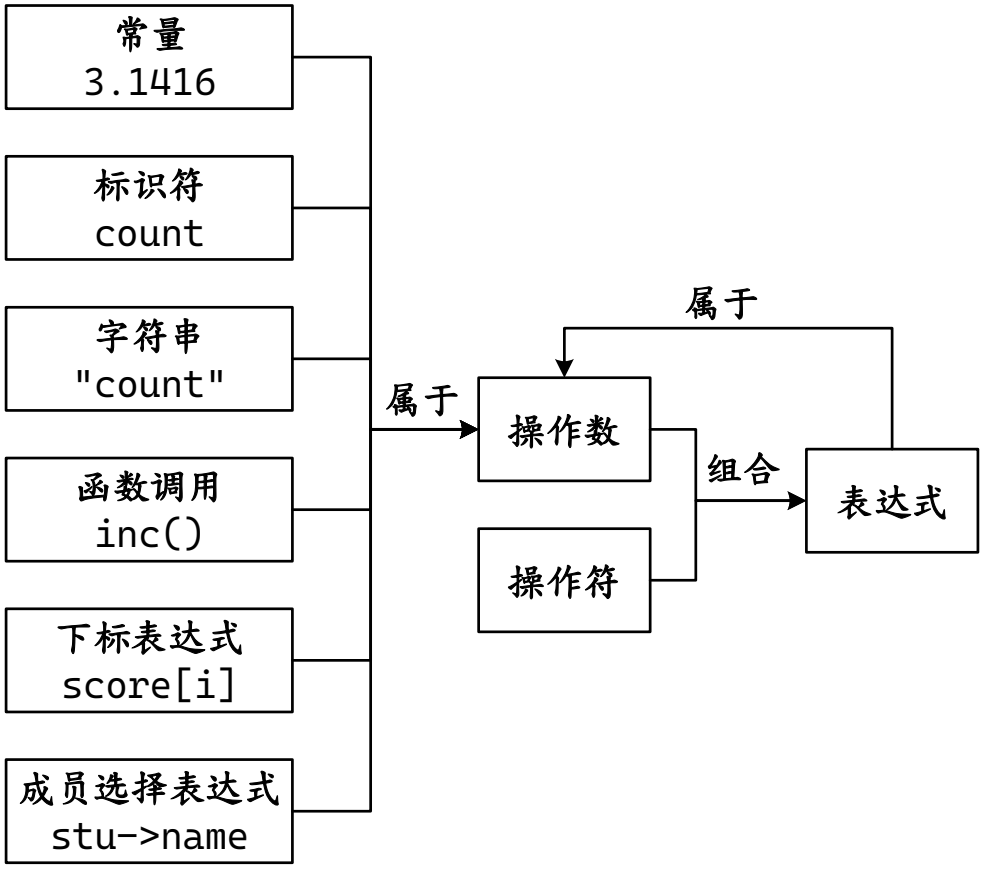
```
Shoe size (men's)    foot length
       3.0              8.31 inches
       4.0              8.64 inches
（此处忽略一部分）
      18.0             13.30 inches
If the shoe fits, wear it.
```

目录

运算符和操作数表达式

- 表达式是一系列的运算符和操作数的任意组合
- 表达式的主要作用
  - 计算值
  - 指定对象或函数
- 表达式的副作用
  - 可能改变指定对象的值

表达式	值	改变值
-4+6	2	×
c=3+8	11	√
6+(c=3+8)	17	√



运算符分类

- 运算符按操作数不同，分为一元、二元、三元

分类	格式	示例
一元	<运算符><操作数>	-16
二元	<操作数1><运算符><操作数2>	5+3
三元	<操作数1><运算符1><操作数2><运算符2><操作数3>	h?16:2

运算符分类

- C语言的所有运算符

分类	运算符
一元	后缀增量（++）、后缀减量（--）、函数调用（()）、数组下标（[]）、结构/联合成员（.）、结构/联合指针成员（->）、复合文字（{list}）、前缀增量（++）、前缀减量（--）、正号（+）、负号（-）、逻辑非（!）、按位非（~）、强制类型转换（(type)）、间接寻址（*）、取址（&）、存储空间（sizeof）
二元	乘（*）、除（/）、模（%）、加（+）、减（-）、左移位（<<）、右移位（>>）、大于（>）、小于（<）、大于等于（>=）、小于等于（<=）、相等（==）、不相等（!=）、按位与（&）、按位异或（^）、按位或（|）、逻辑与（&&）、逻辑或（||）、赋值（=）、加法赋值（+=）、减法赋值（-=）、乘法赋值（*=）、除法赋值（/=）、取模赋值（%=）、左移赋值（<<=）、右移赋值（>>=）、按位与赋值（&=）、按位异或赋值（^=）、按位或赋值（|=）、逗号（,）
三元	条件运算符（?:）

目录

算术运算符（+−*/%）

算术运算

格式	*exp1* + *exp2*	计算两数相加之和。
	exp1 − *exp2*	计算两数相减之差。
	exp1 * *exp2*	计算两数相乘之积。
	exp1 / *exp2*	计算两数相除之商。
	exp1 % *exp2*	计算两个整数相除之余，称“模”。
求值	计算结果，类型为两个操作数数据类型的较高优先级	
示例	`32 / 5.0`	整体值为6.4
说明	1. 和小学的知识基本一致。 2. 特殊情况包括：两个操作数都是整数的除法，执行整除，否则执行实数除法；求模时，两个操作数仅限整数。 3. 无副作用。	

算术运算符

- 算术运算的四则运算规则与算术课基本相同
- 参与四则运算的数据类型限定了范围和精度
- 运算的类型以两个操作数数据类型较高优先级为准
  - 低优先级的操作数先转换为该类型再计算，结果为该类型

优先级顺序	long double	double	float	long long int	int	short	char

示例	两个操作数级别	值
-7.3L+0.3f	long double √, int	-6.999999988079
-7.3f+4.0	float, double √	1.75
-1u+3L	int, long long int √	4294967298
(short)65535*(char)3	short √, char	-3（溢出）

```c
/* squares.c -- produces a table of first 20 squares */
#include <stdio.h>
int main(void)
{
    int num = 1;

    while (num < 21)
    {
        printf("%4d %6d\n", num, num * num);
        num = num + 1;
    }

    return 0;
}
```

```
   1      1
   2      4
   3      9
（此处忽略一部分）
  20    400
```

算术运算符

- 操作数都是整数的除法，执行整除
  - 商和余数的值按被除数和除数绝对值相除
  - 商的符号按被除数与除数是否同号，余数符号同被除数

int X	int Y	X/Y	X%Y
7	4	1	3
-7	4	-1	-3
7	-4	-1	3
-7	-4	1	-3

X	Y	X/Y
7	4	1
7.0	4	1.75
7	4.0	1.75
7.0	4.0	1.75

- 取模（余数）的操作数仅限整数，否则出现编译错误

```
[Error] invalid operands to binary % (have 'double' and 'int')
```

```c
/* divide.c -- divisions we have known */
#include <stdio.h>
int main(void)
{
    printf("integer division:  5/4   is %d \n", 5/4);
    printf("integer division:  6/3   is %d \n", 6/3);
    printf("integer division:  7/4   is %d \n", 7/4);
    printf("floating division: 7./4. is %1.2f \n", 7./4.);
    printf("mixed division:    7./4  is %1.2f \n", 7./4);

    return 0;
}
```

这里的1不是整数部分长度，而是最小宽度，实际上，数字宽度最小是1，因而被忽略

7 是整型常量，
7.是双精度型常量

```
integer division:  5/4   is 1
integer division:  6/3   is 2
integer division:  7/4   is 1
floating division: 7./4. is 1.75
mixed division:    7./4  is 1.75
```

```c
/* convert.c -- automatic type conversions */
#include <stdio.h>
int main(void)
{
    char ch;
    int i;
    float fl;
    fl = i = ch = 'C';
    printf("ch = %c, i = %d, fl = %2.2f\n", ch, i, fl);
    ch = ch + 1;
    i = fl + 2 * ch;
    fl = 2.0 * ch + i;
    printf("ch = %c, i = %d, fl = %2.2f\n", ch, i, fl);
    ch = 1107;
    printf("Now ch = %c\n", ch);
    ch = 80.89;
    printf("Now ch = %c\n", ch);
    return 0;
}
```

```
convert.c: In function ‘main’:
convert.c:15:5: warning: overflow in implicit
constant conversion [-Woverflow]
     ch = 1107;
     ^
```

```
ch = C, i = 67, fl = 67.00
ch = D, i = 203, fl = 339.00
Now ch = S
Now ch = P
```

麦子与国王问题

- **题目**

  相传古印度舍罕王的宰相达依尔是国际象棋的发明者。国王因其贡献，问他想要什么奖励。达依尔说：“请在国际象棋棋盘上按规律摆上麦子：第1格1粒，第2格2粒，……，后面一格的麦子总是前一格麦子数的两倍，摆满整个棋盘赐我，我就感恩不尽了。”国王一想，这还不容易，就答应他了。其实，整个印度全国的麦子都给他也不够！

- **归纳规律**

$$n = \sum_{i=1}^{64} 2^{i-1}$$

```
total = current = 1.0;
while (count < SQUARES)
{
    count = count + 1;
    current = 2.0 * current;
    total = total + current;
}
```

```c
/* wheat.c -- exponential growth */
#include <stdio.h>
#define SQUARES 64              // squares on a checkerboard
int main(void) {
    const double CROP = 2E16;  // world wheat production
in wheat grains
    double current, total;
    int count = 1;
    printf("square     grains
    printf("fraction of \n");
    printf("            added
    printf("world total\n");
    total = current = 1.0; /* start with one grain  */
    printf("%4d %13.2e %12.2e %12.2e\n", count, current,
total, total/CROP);
    while (count < SQUARES) {
        count = count + 1;
        current = 2.0 * current; /* double grains on next square */
```

进入循环前，初始状态很重要。如乘法的初始值为0将出错。有经验的程序员不会在编程之处纠结该值，往往先赋一个值，再通过测试去确定该值。

循环的执行条件

循环条件的变化（否则可能死循环）

```
        total = total + current;      /* update total */
        printf("%4d %13.2e %12.2e %12.2e\n", count, current,
               total, total/CROP);
    }
    printf("That's all.\n");
    return 0;
}
```

相同内容的并排输出要对齐，增加可读性。一般通过格式字符串中的宽度来控制。

```
square     grains        total      fraction of
           added         grains     world total
   1     1.00e+000     1.00e+000     5.00e-017
   2     2.00e+000     3.00e+000     1.50e-016
（此处忽略一部分）
  64     9.22e+018     1.84e+019     9.22e+002
That's all.
```

继续循环，将出现溢出

```
 126     4.25e+037     8.51e+037     4.25e+021
 127     8.51e+037     1.70e+038     8.51e+021
 128     1.70e+038     1.#Je+000     1.#Je+000
 129     1.#Je+000     1.#Je+000     1.#Je+000
```

1.#INF0e+000 配合 %13.2e 输出 1.#Je+000

目录

赋值运算符

赋值

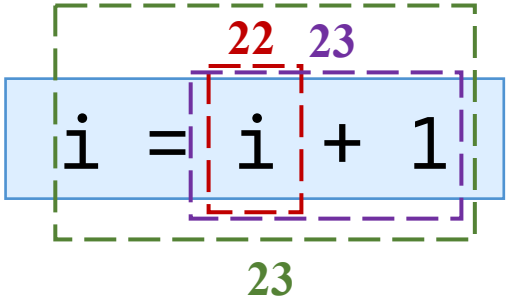
功能	计算右值，做为左值的新值	
格式	*lvalue* = *rvalue*	
参数	*lvalue*	左值，限于：变量名、数组元素、结构体成员
	rvalue	右值，可以是表达式。
求值	右值的值	
副作用	改变了左值的值	
示例	x = 32 + 5	整体值为37，运行后x值为37
说明	A = B 不表示“A等于B”	

赋值运算符

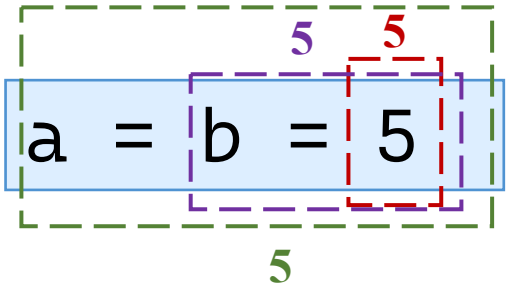
- 左值超出限定，则判为编译错误
  - 左值不得为常量或常数

```
2018 = a;
```

- 右值表达式含有左值，按旧值计算再改变值
  - 先按当前变量的值计算右值，得到表达式的值
  - 用右值改变左值，作为左值的新值
  - 例：原i值为22，运行后i新值为23



- 赋值表达式嵌套形成多重复值
  - 从右往左执行赋值



```c
/* golf.c -- golf tournament scorecard */
#include <stdio.h>
int main(void)
{
    int jane, tarzan, cheeta;

    cheeta = tarzan = jane = 68;
    printf("                        cheeta   tarzan    jane\n");
    printf("First round score %4d %8d %8d\n",cheeta,tarzan,jane);

    return 0;
}
```

等价于：

```c
jane=68;
tarzan=jane;
cheeta=tarzan;
```

```
                        cheeta   tarzan    jane
First round score         68       68       68
```

复合赋值

复合赋值

格式	*lvalue* += *rvalue*	计算*lvalue* = *lvalue* + *rvalue*。
	lvalue -= *rvalue*	计算*lvalue* = *lvalue* - *rvalue*。
	lvalue *= *rvalue*	计算*lvalue* = *lvalue* * *rvalue*。
	lvalue /= *rvalue*	计算*lvalue* = *lvalue* / *rvalue*。
	lvalue %= *rvalue*	计算*lvalue* = *lvalue* % *rvalue*。
求值	计算*lvalue op rvalue*的值	
副作用	用*lvalue op rvalue*的值改变*lvalue*的值。	
示例	c /= 5.0	设运行前c为6.4，整体值为6.4 运行后，c的值为6.4
说明	1. 运算规则（尤其除、模等），和其等价的展开式相同。 2. *lvalue*的左值规则，与普通赋值符号的规则相同。	

复合赋值

- 复合赋值与其展开式
  - 书写程序时使用复合赋值替代其展开式，显得更加简洁
  - 预估复合赋值的运算结果时，用其展开式代替
- 左值和右值的约束条件，与其展开式的约束相同。
  - 左值限于：变量名、数组元素、结构体成员
  - 除法赋值：左值右值都是整型时，整除；否则，实数除
  - 取模赋值：限于左值和右值是整型

表达式（int n; double d;）	运行
n /= 3;	执行整除
d /= 3;	执行实数除法
n %= 3;	执行取模

表达式	运行
(d + 5) /= 3;	编译错误
d %= 3;	编译错误
3 /= 2;	编译错误

目录

强制类型转换

强制类型转换

功能	将操作数强制转换为新类型	
格式	(type)exp1	
参数	type	类型名，需要强制转换的目标类型
	exp1	待转换的操作数
求值	强制类型转换后的值	
示例	(int)32.5	整体值为32
说明	强制类型转换可能导致精度和范围截断	

强制类型转换

- 强制类型转换与自动类型转换的区别联系
  - 自动类型转换通常是从数据范围较小的类型自动转换为数据范围较大的类型，不会丢失精度和范围
  - 强制类型转换可能会丢失精度或范围
- 如不强制类型转换结果也正确，不必强制类型转换

表达式语句（mice为double类型）	运行后mice的值	说明
`mice = 1.6 + 1.7 + 1.8;`	5.1	直接相加。
`mice = (int)1.6 + 1.7 + (int)1.8;`	3.7	第1、3项值为1。
`mice = (int)(1.6 + 1.7) + (int)1.8;`	4.0	第1项值为3。
`mice = int(1.6 + 1.7) + int(1.8);`	编译错误	这是C++的语法，不是C的语法。

尺寸运算符

尺寸

功能	将操作数强制转换为新类型	
格式	sizeof(*type*) 或 sizeof(*exp*)或 sizeof *exp*	
参数	*type*	类型名，需要求尺寸的类型，该项括号是必须的
	exp	表达式，需要求尺寸的表达式
求值	计算类型的尺寸，或表达式对应类型的尺寸，计算结果对应数据类型为size_t	
示例	sizeof(double)	整体值为8
说明	1. 表达式不会被计算，表达式的副作用不会生效。 2. 不同程序架构和编译器下，一些类型的值结果可能不同。	

尺寸运算符

- 注意事项
  - 尺寸运算符后的表达式，不计算，副作用也不生效

```
int a = 4;
int s = sizeof(a=3);
```

此处a=3不会生效，仅计算a=3对应类型的尺寸。运行后a为4，s为4。

  - 同一类型的尺寸与应用程序架构有关
    - 在GCC 64位程序中，sizeof(long)是8，一般情况是4。
  - size_t类型尺寸与应用程序架构有关
    - 在32位应用程序中，size_t是32位无符号整数
    - 在64位应用程序中，size_t是64位无符号整数

```c
// sizeof.c -- uses sizeof operator
// uses C99 %z modifier -- try %u or %lu if you lack %zd
#include <stdio.h>
int main(void)
{
    int n = 0;
    size_t intsize;

    intsize = sizeof (int);
    printf("n = %d, n has %zd bytes; all ints have %zd bytes.\n",
           n, sizeof n, intsize );

    return 0;
}
```

有经验的程序员不会对同一类量使用不同的实现风格。

```
n = 0, n has 4 bytes; all ints have 4 bytes.
```

```c
// min_sec.c -- converts seconds to minutes and seconds
#include <stdio.h>
#define SEC_PER_MIN 60              // seconds in a minute
int main(void) {
    int sec, min, left;
    printf("Convert seconds to minutes and seconds!\n");
    printf("Enter the number of seconds (<=0 to quit):\n");
    scanf("%d", &sec);              // read number of seconds
    while (sec > 0) {
        min = sec / SEC_PER_MIN; // truncated number of minutes
        left = sec % SEC_PER_MIN; // number of seconds left over
        printf("%d seconds is %d minutes, %d seconds.\n", sec,
               min, left);
        printf("Enter next value (<=0 to quit):\n");
        scanf("%d", &sec)
    }
    printf("Done!\n");
    return 0;
}
```

```
Convert seconds to minutes and seconds!
Enter the number of seconds (<=0 to quit):
255↲
255 seconds is 4 minutes, 15 seconds.
Enter next value (<=0 to quit):
-3↲
Done!
```

目录

增量减量运算符

增量减量

格式	*exp1*++	先对*exp1*取值，再对*exp1*自增1
	++*exp1*	先对*exp1*自增1，再对*exp1*取值
	exp1--	先对*exp1*取值，再对*exp1*自减1
	--*exp1*	先对*exp1*自减1，再对*exp1*取值
求值	前缀增减量，副作用先生效，再取操作数的值； 后缀增减量，先取操作数的值，副作用再生效。	
副作用	增量运算符，*exp1*自增1；减量运算符，*exp1*自减1。	
示例	c++;	设运行前c为6，整体值为6，运行后，c的值为7
	--c;	设运行前c为6，整体值为5，运行后，c的值为5
说明	1. 注意区分表达式的值和运行前后左值的值。 2. *exp1*应是左值。	

增量减量运算符

- 前缀和后缀增减量的区别
  - 前缀先增减量再求值；后缀先求值再增减量

表达式的值和exp1的值不是一回事

名称	表达式	表达式的值	运行后exp1的值
前缀增量运算符	++exp1	exp1+1的值	exp1+1的值
后缀增量运算符	exp1++	exp1的值	exp1+1的值
前缀减量运算符	--exp1	exp1-1的值	exp1-1的值
后缀减量运算符	exp1--	exp1的值	exp1-1的值

- 程序员编程时对增量减量的使用应适可而止
  - 程序员有义务让读者看懂（除非你要保护技术秘密）
  - 适当合并表达式简化代码，但过度合并难以阅读

```
ans /= num / 2 + 6 * ( 1 + num++ + ++num ); // ans=7, num=3
```

```c
/* add_one.c -- incrementing: prefix and postfix */
#include <stdio.h>
int main(void)
{
    int ultra = 0, super = 0;

    while (super < 5)
    {
        super++;
        ++ultra;
        printf("super = %d, ultra = %d \n", super, ultra);
    }

    return 0;
}
```

```
super = 1, ultra = 1
super = 2, ultra = 2
super = 3, ultra = 3
super = 4, ultra = 4
super = 5, ultra = 5
```

```c
/* post_pre.c -- postfix vs prefix */
#include <stdio.h>
int main(void) {
    int a = 1, b = 1;
    int a_post, pre_b;

    a_post = a++;  // value of a++ during assignment phase
    pre_b = ++b;   // value of ++b during assignment phase
    printf("a  a_post   b   pre_b \n");
    printf("%1d %5d %5d %5d\n", a, a_post, b, pre_b);

    return 0;
}
```

a的值是1，所以a++的值是1，a_post的值是1，运行后a的值是2

b的值是1，所以++b的值是2，pre_b的值是2，运行后b是2

```
a  a_post   b   pre_b
2     1     2     2
```

增量减量运算符

- 增量减量运算符与循环语句混合使用可控制循环次数

```
int i = 0;              // initialized value
while (i < 5)           // loop condition
{
   // statements here ...
   i++;                 // count 1 by 1 in loop
}
```

  - 等价于（for循环）

```
for (i = 0; i < 5; i++) // init. value; loop condition; count
{
    // statements here ...
}
```

```c
/* bottles.c */
#include <stdio.h>
#define MAX 100
int main(void) {
    int count = MAX + 1;
    while (--count > 0) {
        printf("%d bottles of spring water on the wall, "
               "%d bottles of spring water!\n", count, count);
        printf("Take one down and pass it around,\n");
        printf("%d bottles of spring water!\n\n", count - 1);
    }
    return 0;
}
```

```
（此处省略数行）
2 bottles of spring water on the wall, 2 bottles of spring water!
Take one down and pass it around,
1 bottles of spring water!

1 bottles of spring water on the wall, 1 bottles of spring water!
Take one down and pass it around,
0 bottles of spring water!
```

目录

关系运算符

关系运算

格式	*exp1* < *exp2*	判断*exp1*是否小于*exp2*
	exp1 <= *exp2*	判断*exp1*是否小于或等于*exp2*
	exp1 < *exp2*	判断*exp1*是否大于*exp2*
	exp1 <= *exp2*	判断*exp1*是否大于或等于*exp2*
	exp1 == *exp2*	判断*exp1*是否等于*exp2*
	exp1 != *exp2*	判断*exp1*是否不等于*exp2*
求值	判断*exp1*和*exp2*的具体关系。为真，值为1；为假，值为0。	
示例	3 >= 3	值为1
说明	1. 大（或小）于（等）于号的运算优先级高于（不）等号。 2. 浮点数存在舍入误差，慎用相等运算符。	

浮点数的相等

- 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
```

$$1.09999990463256835937_5 = 1\times2^0+1\times2^{-4}+1\times2^{-5}+1\times2^{-8}+1\times2^{-9}+1\times2^{-12}+1\times2^{-13}+1\times2^{-16}+1\times2^{-17}+1\times2^{-20}+1\times2^{-21}$$

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

$$0.09999999403953552246093_75, 0.10000000149011611938476_5625$$

```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;
}
```

通过这个程序可以查看真和假在参与运算时的值。

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

```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);

    return 0;
}
```

如果此处错写为 status = 1 的值恒为1，则为死循环。在输入不是整数时无法退出。

```
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。

有经验的程序员会把这句话写在while的条件里。

目录

逻辑运算符

逻辑与或

格式	*exp1* && *exp2*	判断*exp1*和*exp2*是否同时为真
	exp1 || *exp2*	判断*exp1*和*exp2*是否不同时为假
求值	判断*exp1*和*exp2*的逻辑关系。 非零与非零为1，其余为0；零或零为0，其余为1。	
示例	3 && 3	值为1
	5 && 0	值为0
	0 || 6	值为1
	0 || 0	值为0
说明	1. 参与逻辑运算的两个表达式，非0表示真，0表示假。 2. 计算结果，真为1，假为0。 3. 短路计算：*exp1*为假时，逻辑与的*exp2*不会被计算； *exp1*为真时，逻辑或的*exp2*不会被计算。	

逻辑运算符

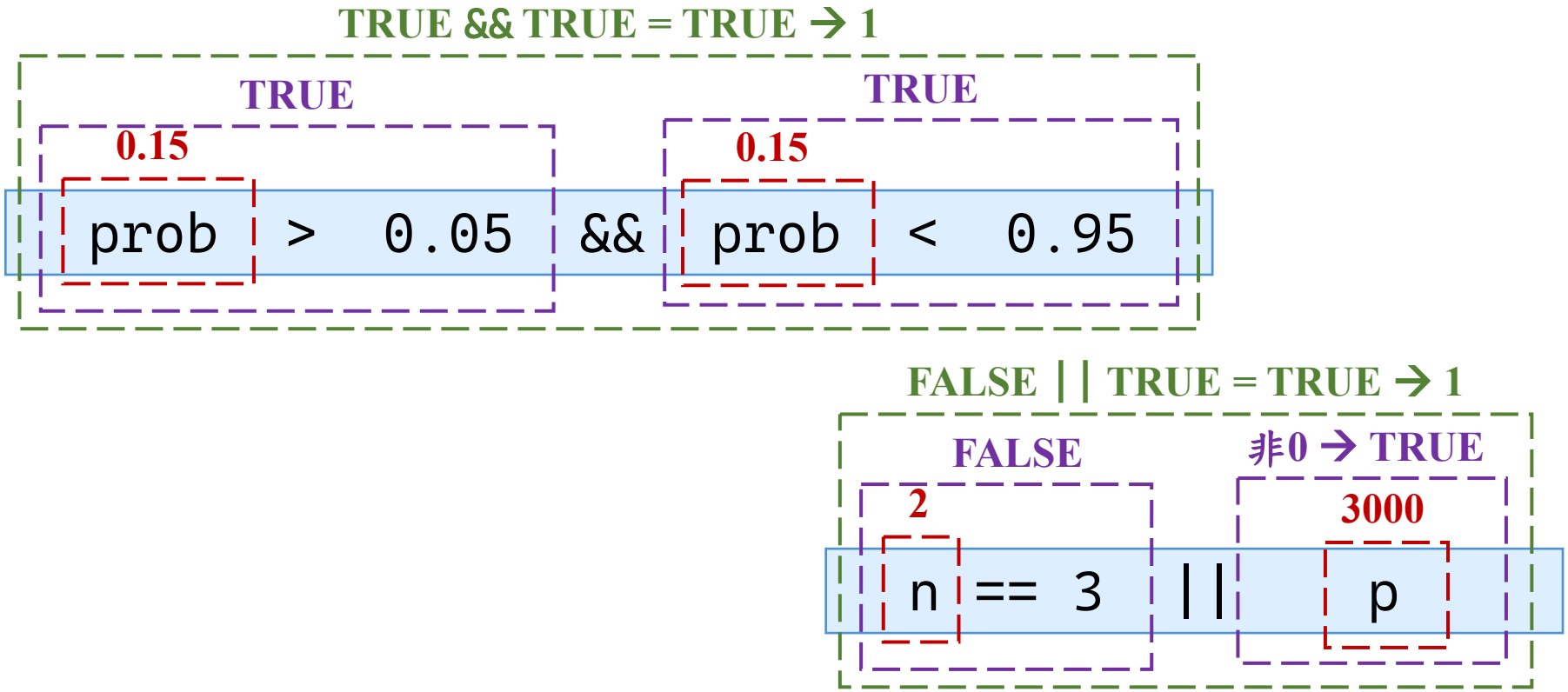
- 逻辑与相当于算术乘法，逻辑或相当于算术加法

```
1 && 1 → 1
0 && 1 → 0
1 && 0 → 0
0 && 0 → 0
```

```
1 || 1 → 1
0 || 1 → 1
1 || 0 → 1
0 || 0 → 0
```

逻辑表达式的真假值

- 逻辑表达式的结果：真为1，假为0
- 逻辑表达式的输入：以“非零”为真，以0为假



```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
```

逻辑表达式的短路计算

- 左侧表达式为假时，逻辑与的右侧表达式不会被计算
  - 假与任何数都为假，该情况右侧表达式不需要计算
- 左侧表达式为真时，逻辑或的右侧表达式不会被计算
  - 真或任何数都为真，该情况右侧表达式不需要计算

名称	运算符	伪代码描述	短路计算规则
逻辑与	x && y	if (x) y else FALSE	x等于0时，y不计算。
逻辑或	x || y	if (x) TRUE else y	x不等于0时，y不计算。

FALSE && ANY = FALSE → 0

0.05 FALSE

```
p > 0.05 && ++p < 0.95
```

不被计算

TRUE || ANY = TRUE → 1

3000 TRUE

```
p || p++ < 0.95
```

不被计算

```c
// chcount.c  -- use the logical AND operator
#include <stdio.h>
#define PERIOD '.'
int main(void)
{
    char ch;
    int charcount = 0;

    while ((ch = getchar()) != PERIOD)
    {
        if (ch != '"' && ch != '\'')
            charcount++;
    }
    printf("There are %d non-quote characters.\n",
charcount);

    return 0;
}
```

注意单个变量在搭配不等号时配合“与”，反之在搭配等号时配合“或”

```
I'm here. Where?↲
There are 7 non-quote characters.
```

取值范围的判断

- 用与和或来表示范围（相当于集合的交集和并集）
  - `range>90 && range<=100` 表示 $\text{range} \in (90, 100]$
  - `range>90 || range<=80` 表示 $\text{range} \in (-\infty, 80] \cup (90, +\infty)$
  - 错误的表达式

```
90<range<=100
```

    - 根据优先级顺序先算`90<range`，结果为0或1
    - 再计算`0<=100`或`1<=100`，结果一定为1

逻辑运算符

逻辑否

格式	!exp1	判断exp1是否为假
求值	判断exp1是否为0。exp1为0，则值为1；exp1非0，则值为0。	
示例	!3	值为0
	!0	值为1
说明	1. 参与逻辑运算的表达式，非0表示真，0表示假。 2. 计算结果，真为1，假为0。	

目录

逗号运算符

逗号

格式	*exp1*，*exp2*	计算*exp1*和*exp2*的值
求值	表达式的值为*exp2*的值。	
示例	2，1	值为1
	x = 1，3	值为3
说明	无。	

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;   // x is assigned 2, and value expression is 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
```

目录

条件运算符

条件运算符

格式	*exp1*?*exp2*:*exp3*	根据*exp1*的值计算*exp2*或*exp3*的值
求值	如果*exp1*的值非0，则表达式取*exp2*的值，且*exp3*不被计算。 如果*exp1*的值为0，则表达式取*exp3*的值，且*exp2*不被计算。	
示例	(x == 2)?4:0	如果运行前x值为1，则表达式值为0
	(x == 2)?4:0	如果运行前x值为2，则表达式值为4
说明	1. 表达式(*x*)?*y*:*z*相当于if (*x*) *y* else *z*。 2. 无关的表达式不会被计算。	

条件运算符

- 条件表达式格式 <表达式1>?<表达式2>:<表达式3>
- 表达式的短路

```
(x)?y:z 相当于 if (x) y else z
```

  - 当x为真，取y的值， z不被计算
  - 当x为假，取z的值， y不被计算

- 条件运算符用于组成表达式，if-else组成语句

```
int max(int a, int b) {
    return (a > b) ? a : b;
}
```

例题：一个喷漆程序

- 已知每罐油漆可喷的面积为350平方英尺。对任意输入的面积（整数），推出需要的油漆罐数。
- 思路
  - 循环scanf接受输入%d，计算350的倍数，上取整。
  - 使用条件运算符，根据罐数判断是否加复数s。

```
Enter number of square feet to be painted:
350↲
You need 1 can of paint.
Enter next value (q to quit):
701↲
You need 3 cans of paint.
Enter next value (q to quit):
```

```c
/* paint.c -- uses conditional operator */
#include <stdio.h>
#define COVERAGE 350        // square feet per paint can
int main(void)
{
    int sq_feet;
    int cans;
    printf("Enter number of square feet to be painted:\n");
    while (scanf("%d", &sq_feet) == 1)
    {
        cans = sq_feet / COVERAGE;
        cans += ((sq_feet % COVERAGE == 0)) ? 0 : 1;
        printf("You need %d %s of paint.\n", cans,
               cans == 1 ? "can" : "cans");
        printf("Enter next value (q to quit):\n");
    }
    return 0;
}
```

该部分可简写为
(sq_feet % COVERAGE != 0)

条件表达式中的表达式部分，可以为字符串或其它表达式。

目录

表达式的作用

- 主要作用：求值
- 副作用：修改数据对象

代码	主要作用（求值）	副作用（改变对象的值）
`length = 5`	表达式值为5	修改length的值为5
`sqrt(--length)`	表达式值为2	修改length的值为4
`count >= 3`	表达式值为1 （设count值为3）	无

表达式求值

- 运算符的优先级：规定运算结合顺序
- 表达式的序列点：规定副作用顺序
  - 规定表达式中副作用的发生顺序，用于规避结果不确定性。
  - 序列点前的副作用已经发生，点后的副作用尚未发生。
  - 两个序列点之间的多个副作用，其顺序不确定。

虽然本式优先级为：除法、加法、逻辑或、赋值，但逻辑或是序列点，逻辑或的结合顺序中，序列点前先算，故加法先算，再视情况算除法，逻辑或，最后赋值。

```
ans = num + 2 || num / 5 ;
```

运算符的优先级

- 运算符的优先级：规定运算结合顺序
  - 规定表达式含多种运算时的结合顺序，保证运算结果唯一。
  - 优先级不总是表示运算符的计算顺序。
    - 甚至有的部分被短路，不会生效

表达式	结合顺序
x+1>3 && y-1<4	(((x+1)>3) && ((y-1)<4))
sqrt((u-x)*(u-x)+(v-y)*(v-y))	(sqrt((u-x)*(u-x)+(v-y)*(v-y)))
x=i++?(j*=5):(j*=3);	(x=((i++)?(j*=5):(j*=3)));
(int)d%7+(int)d/7	(((int)d)%7)+(((int)d)/7))

运算符的优先级

- 下表为与本节相关的优先级顺序
  - 一元<二元<三元<赋值<逗号，算术<关系<逻辑

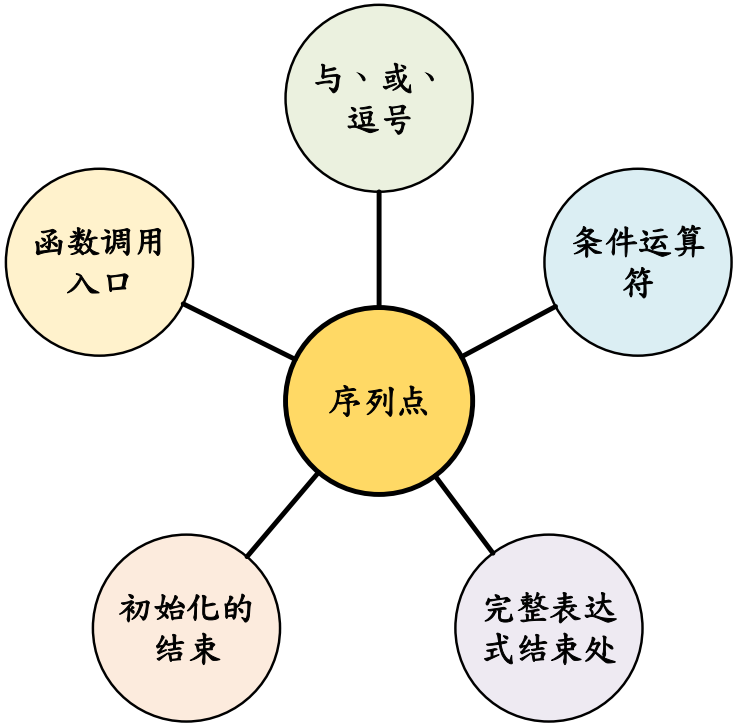
序号	符号	说明	序号	符号	说明
1 →	后缀++ --	后缀增减量	4	+ -	算术运算：加减
	( )	函数调用	6	< > <= >=	关系运算符：大小
	[ ]	数组下标	7	== !=	关系运算符：相等
2 ←	前缀++ --	前缀增减量	11	&&	逻辑运算符：与
	+ -	正负号	12	||	逻辑运算符：或
	!	逻辑运算符：非	13	?:	三元条件运算符
	(type)	强制类型转换	14 ←	=	赋值
	sizeof	存储空间		+= -= *= /= %=	复合运算符
3	* / %	算术运算：乘除模	15	,	逗号表达式

序列点

- 序列点（sequence points）是计算机程序的执行点
  - 要求该点前的副作用已经发生，该点后副作用尚未开始
  - 两个序列点间的多个副作用，其执行顺序不确定。
    - 程序员应该避免未定义行为。

```
for (i=0, j=++i + i++; i<=j; i++)
```

在 VS 中，i为2，j为2；
在GCC中，i为2，j为3。



序列点

- 函数调用时的函数入口点。
  - 计算函数的所有参数，并在输入函数前完成所有副作用。
  - 但未指定参数之间的计算顺序。
    - 常见的编译器，计算顺序为从右至左，但C标准未定义

```
#include <stdio.h>
void f(int i, int j) {
    printf("i = %2d, j = %2d.\n", i, j);
}
int main() {
    int i = 1;
    f(i += 3, i *= 5);
    return 0;
}
```

```
i =  8, j =  8.
```

先结算i*=5，得到i为5；
再结算i+=3，得到i为8。
也可以反过来，此时i,j为20。

序列点

- 逻辑与、或、逗号的左与右操作数求值之间
  - 逻辑与和逻辑或是短路求值的一部分
    - 逻辑与左值为0时，逻辑或左值不为0时，右值不会计算

```c
#include <stdio.h>
int main() {
    int i = 2, j = 3, x, y, z;
    x = (i -= 2) && (j *= 5);
    printf("x = %2d, i = %2d, j = %2d.\n", x, i, j);
    y = (j++) || (j *= 5);
    printf("y = %2d, i = %2d, j = %2d.\n", y, i, j);
    z = (i += 3, j *= 5);
    printf("z = %2d, i = %2d, j = %2d.\n", z, i, j);
    return 0;
}
```

```
x =  0, i =  0, j =  3.
y =  1, i =  0, j =  4.
z = 20, i =  3, j = 20.
```

序列点

- 条件运算符的第一操作数后，第二或三操作数前

```
#include <stdio.h>
int main() {
    int i = -1, j = 3, x;
    x = (i++) ? (j *= 5) : (j *= 3);
    printf("x = %2d, i = %2d, j = %2d.\n", x, i, j);
    return 0;
}
```

```
x = 15, i =  0, j = 15.
```

- 初始化的结束

```
#include <stdio.h>
int main() {
    int i = 1, j = (i+=4);
    printf("i = %2d, j = %2d.\n", i, j);
    return 0;
}
```

```
i =  5, j =  5.
```

序列点

- 完整表达式结束处
  - 表达式语句、返回语句，if、switch、while、do-while语句的控制表达式，for语句的3个表达式。

```c
#include <stdio.h>
int f(int i) {
    return i++;
}
int main() {
    int i = 2, j = 3;
    printf("i =  %2d, f(i) =  %2d.\n", i, f(i));
    if (i == 2)
        printf("# i =  %2d, j =  %2d.\n", i, j);
    for (int i = (j /= 2); i < j + 6; i += 3)
        printf("* i =  %2d, j =  %2d.\n", i, j);
    return 0;
}
```

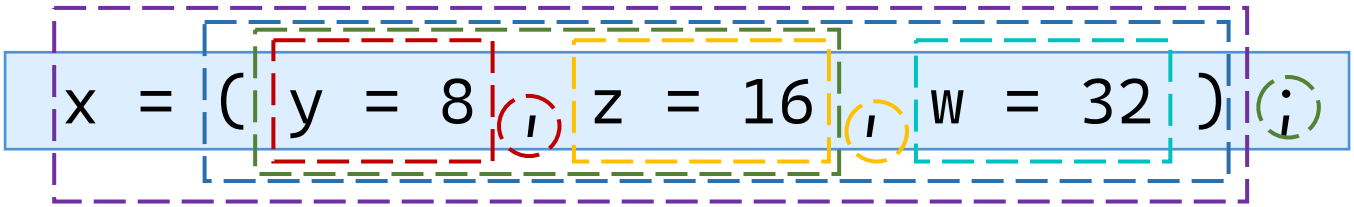
```
i =  2, f(i) =  2.
# i =  2, j =  3.
* i =  1, j =  1.
* i =  4, j =  0.
```

```c
/* rules.c -- precedence test */
#include <stdio.h>

int main(void)
{
    int top, score;

    top = score = -(2 + 5) * 6 + (4 + 3 * (2 + 3));
    printf("top = %d, score = %d\n", top, score);

    return 0;
}
```

```
top = -23, score = -23
```

表达式的计算

- 计算顺序
  - 按优先级将表达式划分出子表达式的内部结构
  - 找出序列点，将所在子表达式划分成多个子表达式
  - 序列点划分的子表达式按优先级顺序计算，副作用生效



先划分子表达式，, 决定先算 再算 ，, 决定先算 再算 ，最后，; 决定算 ，序列点间先算 再算 。

序列点 , 优先级 [ ] 优先顺序 

表达式的计算

- 计算表达式的值应脚踏实地，跳步将造成混乱

```
int x, y, z, w;
x = (y = 8, z = 16, w = 32 );
y = !z || w > z > y;
z = x == w;
w += w /= w + 1;
```

x	y	z	w
32	8	16	32
32	0	16	32
32	0	1	32
32	0	1	0

```
x=(8,16,w=32) → x=(16,32)   → x=32    → 32
y=0||32>16>8  → y=0||1>8    → y=0||0  → y=0 → 0
z=32==32;     → z=1         → 1
w+=w/=32+1    → w+=w=32/33  → w=0+0   → 0
```

目录

语句

- C语言语句的分类

语句

声明语句	`int x, y, z;`
标签语句	`stop: printf("Stop!\n");`
复合语句	`{ x=5; y=6; }`
表达式语句	`y=0.3*x+6+eps;`
循环语句	`while (x<3) x++;`
选择语句	`if (x>5) y=1; else y=4;`
跳转语句	`if (x==0) break;`
空语句	`;`
返回语句	`return 0;`

语句示例

- 声明语句
  - 用于声明变量、函数或类型等

```
int count;
int inc(int a);
```

- 标签语句
  - 由 标签、冒号、语句 构成
    - switch语句的case部分是标签语句
  - 通过跳转语句实现跳转

```
    if (i == 5)
        goto stop;
  …
stop:
    printf("Stop.\n");
```

- 表达式语句
  - 由表达式加上分号构成的语句

```
y = sqrt(x);
length = 5;
```

语句示例

- 复合语句
  - 以花括号为界，自上而下执行
  - 构成另一个语句的组成部分的语句称为封闭语句的“体”。
    - 如：函数体、循环体等

```
{
    line[i] = x;
    x*=2;
    i++;
}
```

- 选择语句
  - if、if-else语句
  - switch-case语句

```
if ( i > 0 )
    y = x / i;
else
    y = f( x );
```

语句示例

- 循环语句
  - for、while、do-while语句
- 跳转语句
  - break语句、continue语句
- 空语句
  - 没有内容的语句
- 返回语句
  - 以return为关键字的语句
  - 用于跳出当前函数

```
for (i=0; i<10; i++)
    line[i] = x;
```

```
for (i=0; i<10; i++) {
    if (x>20)
        break;
    line[i] = x++;
}
```

```
for (i=0; i<10; a[i++]=0)
    ;
```

```
int sum(int i, int j) {
    return i+j;
}
```

```c
/* addemup.c -- five kinds of statements */
#include <stdio.h>
int main(void)        /* finds sum of first 20 integers */
{
    int count, sum;          /* declaration statement    */

    count = 0;               /* assignment statement     */
    sum = 0;                 /* ditto                    */
    while (count++ < 20)     /* while                    */
        sum = sum + count; /*     statement              */
    printf("sum = %d\n", sum);/* function statement      */

    return 0;                /* return statement         */
}
```

```
sum = 210
```

目录

函数

- 将可重复使用的代码封装为函数，变化量为参量

```c
#include <stdio.h>

int main(void)
{
    printf("Shift: %6.3lf.\n", 2.3 * 6 + 1.0 / 2 * 1.2 * 6 * 6);
    printf("Shift: %6.3lf.\n", 3 * 1.5 + 1.0 / 2 * 3 * 1.5 * 1.5);
    printf("Shift: %6.3lf.\n", 1 * 0.8 + 1.0 / 2 * 5 * 0.8 * 0.8);
    return 0;
}
```

```c
double shift(double v0, double a, double t) {
    return v0 * t + 1.0 / 2 * a * t * t;
}
```

```c
printf("Shift: %6.3lf.\n", shift(2.3, 1.2, 6));
printf("Shift: %6.3lf.\n", shift(3, 3, 1.5));
printf("Shift: %6.3lf.\n", shift(1, 5, 0.8));
```

带参数的函数

- 函数需要先声明，有定义，再使用
  - 声明：函数原型

```
double shift(double v0, double a, double t);
```

    - 原型的形式参量(formal parameters)的名字可以省略，但不建议。
    - 如果函数体书写在调用之前，则原型可以省略，但不建议。
  - 定义：函数的实现代码
    - 形式参量：用来接收调用函数时传递的参数。
    - 函数中改变形式参量的值不影响调用该函数的函数中的变量值。

```
double shift(double v0, double a, double t) {
    return v0 * t + 1.0 / 2 * a * t * t;
}
```

```c
/* pound.c -- defines a function with an argument   */
#include <stdio.h>
void pound(int n);  // ANSI function prototype declaration
int main(void) {
    int times = 5;
    char ch = '!';    // ASCII code is 33
    float f = 6.0f;
    pound(times);     // int argument
    pound(ch);        // same as pound((int)ch);
    pound(f);         // same as pound((int)f);
    return 0;
}
void pound(int n) {  // ANSI-style function header
    while (n-- > 0)  // says takes one int argument
        printf("#");
    printf("\n");
}
```

函数的原型声明，一般以全局形式出现

函数的调用。其中，times, ch, f 分别为三次调用时的实际参数。

函数的定义

```
#####
##########################################
######
```

```c
/* pound.c -- defines a function with an argument   */
#include <stdio.h>

void pound(int n) {  // ANSI-style function header
    while (n-- > 0)  // says takes one int argument
        printf("#");
    printf("\n");
}

int main(void) {
    int times = 5;
    char ch = '!';   // ASCII code is 33
    float f = 6.0f;
    pound(times);    // int argument
    pound(ch);       // same as pound((int)ch);
    pound(f);        // same as pound((int)f);
    return 0;
}
```

如果函数体书写在调用之前，则原型可以省略，但不建议。这样容易出错。

```
#####
#####################################
######
```

```c
// running.c -- A useful program for runners
#include <stdio.h>
const int S_PER_M = 60;          // seconds in a minute
const int S_PER_H = 3600;        // seconds in an hour
const double M_PER_K = 0.62137; // miles in a kilometer
int main(void) {
    double distk, distm; // distance run in km and in miles
    double rate;         // average speed in mph
    int min, sec;        // minutes and seconds of running time
    int time;            // running time in seconds only
    double mtime;        // time in seconds for one mile
    int mmin, msec;      // minutes and seconds for one mile
    printf("This program converts your time for a metric race\n");
    printf("speed in miles per hour.\n");
    printf("Please enter, in kilometers, the distance run.\n");
    scanf("%lf", &distk);  // %lf for type double
    printf("Next enter the time in minutes and seconds.\n");
    printf("Begin by entering the minutes.\n");
    scanf("%d", &min);
    printf("Now enter the seconds.\n");
    scanf("%d", &sec);
    // converts time to pure seconds
```

```
    time = S_PER_M * min + sec;
    // converts kilometers to miles
    distm = M_PER_K * distk;
    // miles per sec x sec per hour = mph
    rate = distm / time * S_PER_H;
    // time/distance = time per mile
    mtime = (double) time / distm;
    mmin = (int) mtime / S_PER_M; // find whole minutes
    msec = (int) mtime % S_PER_M; // find remaining seconds
    printf("You ran %1.2f km (%1.2f miles) in %d min, %d sec.\n",
           distk, distm, min, sec);
    printf("That pace corresponds to running a mile in %d min, " mmin);
    printf("%d
    return 0;
}
```

```
This program converts your time for a metric race
speed in miles per hour.
Please enter, in kilometers, the distance run.
231↲
Next enter the time in minutes and seconds.
Begin by entering the minutes.
6↲
Now enter the seconds.
3↲
You ran 231.00 km (143.54 miles) in 6 min, 3 sec.
That pace corresponds to running a mile in 0 min, 2 sec.
Your average speed was 1423.50 mph.
```

目录

C语言程序设计

C Programming

5

谢谢观看

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 炜

(特色化示范性软件学院) 博士,副教授

School of Informatics Wei Huang