# 算术运算

C语言一共有34种运算符，包括了常见的加减乘除运算

## 加法运算+

* 除开能做加法运算，还能表示正号：+5、+90

## 减法运算-

* 除开能做减法运算，还能表示符号：-10、-29

## 乘法运算\*

注意符号，不是x，而是\*

## 除法运算/

* 注意符号，不是÷，也不是\，而是/
* 整数除于整数，还是整数。1/2的值是0，这个并不是二分之一

## 取余运算%

* 什么是取余：两个整数相除之后的余数
* %两侧只能是整数
* 正负性取决于%左侧的数值

## 注意点

1. 自动类型转换

int a = 10.6;

int b = 10.5 + 1.7;

自动将大类型转换为了小类型，会丢失精度

1. 自动类型提升

int b = 10.5 + 10;

将右边的10提升为了double类型

double b = 1.0 / 2;

解决除法的精度问题

1. 强制类型转换

double a = (double)1 / 2;

double b = (double)(1 / 2);

1. 运算顺序

* 表达式
* 结合性（结合方向）:2+3+4
* 优先级：5+4\*8-3

## 习题

1. 当?是什么运算符时，不论整型变量a的值怎么变，结果c都不超过10

int c = a?10;

1. 提示用户输入两个整数，并输出两个整数的平均数
2. 提示用户输入一个时间的秒数，比如500秒就输入500，然后输出对应的分钟和秒，比如500s就是8分钟20秒

# 赋值运算

## 简单赋值

* int a = 10 + 5;的运算过程
* a = b = 10;的运算过程
* 等号左边不能是常量，比如10 = 11;

## 复合赋值

* 复加减乘除余：a += 4 + 5;

# 自增自减

## 简单使用

* ++  自增运算符。如a++，++a，都等价于a = a+1
* 自减运算符。如a--，--a，都等价于a = a-1
* 5++是错误的

## ++a和a++的区别

* int a = 10;
* a++; ++a;
* int b = a++; int b = ++a;

# sizeof

## 作用

用来计算一个变量或者一个常量、一种数据类型所占的内存字节数。

## 基本形式

* sizeof( 变量\常量 )
* sizeof 变量\常量
* sizeof( 数据类型 )
* 不能是sizeof 数据类型

# 关系运算(比较运算)

## 条件判断

* 默认情况下，我们在程序中写的每一句正确代码都会被执行。但很多时候，我们想在某个条件成立的情况下才执行某一段代码



* 这种情况的话可以使用条件语句来完成，但是我们暂时不学习条件语句，先来看一些更基础的知识：如何判断一个条件成不成立。

## 真假

* 在C语言中，条件成立称为“真”，条件不成立称为“假”，因此，判断条件是否成立，就是判断条件的“真假”。
* 怎么判断真假呢？C语言规定，任何数值都有真假性，任何非0值都为“真”，只有0才为“假”。也就是说，108、-18、4.5、-10.5等都是“真”，0则是“假”。

## 关系比较

开发中经常要比较，比如斗地主游戏中牌的大小。利用关系运算符就可以比较两个值的大小。

* 关系运算符的运算结果只有2种：如果条件成立，结果就为1，也就是“真”；如果条件不成立，结果就为0，也就是“假”。

## 使用注意

* 关系运算符中==、!=的优先级相等，<、<=、>、>=的优先级相等，且前者的优先级低于后者：2==3>1
* 关系运算符的结合方向为“从左往右”： 4>3>2
* 关系运算符的优先级小于算术运算符：3+4>8-2

## 习题

计算下列表达式的值

* 3 > 4 + 7
* (3>4) + 7
* 5 != 4 + 2 \* 7 > 3 == 10

# 逻辑运算

* 有时候，我们需要在多个条件同时成立的时候才能执行某段代码，比如：用户只有同时输入了QQ和密码，才能执行登录代码，如果只输入了QQ或者只输入了密码，就不能执行登录代码。这种情况下，我们就要借助于C语言提供的逻辑运算符。
* 逻辑运算的结果只有2个：“真”为1，“假”为0

## 1.&& 逻辑与

1> 使用格式

“条件A && 条件B”

2> 运算结果

只有当条件A和条件B都成立时，结果才为1，也就是“真”；其余情况的结果都为0，也就是“假”。因此，条件A或条件B只要有一个不成立，结果都为0，也就是“假”

3> 运算过程

总是先判断条件A是否成立

* 如果条件A成立，接着再判断条件B是否成立：如果条件B成立，“条件A && 条件B”的结果就为1，即“真”，如果条件B不成立，结果就为0，即“假”
* 如果条件A不成立，就不会再去判断条件B是否成立：因为条件A已经不成立了，不管条件B如何，“条件A && 条件B”的结果肯定是0，也就是“假”

4> 举例

逻辑与的结合方向是“自左至右”。比如表达式 (a>3) && (a<5)

* 若a的值是4：先判断a>3，成立；再判断a<5，也成立。因此结果为1
* 若a的值是2：先判断a>3，不成立，停止判断。因此结果为0
* 因此，如果a的值在(3, 5)这个范围内，结果就为1；否则，结果就为0

5> 注意

* 若想判断a的值是否在(3, 5)范围内，千万不能写成3<a<5，因为关系运算符的结合方向为“从左往右”。 比如a为2，它会先算3<a，也就是3<2，条件不成立，结果为0。再与5比较，即0<5，条件成立，结果为1。因此 3<a<5的结果为1，条件成立，也就是说当a的值为2时，a的值是在(3, 5)范围内的。这明显是不对的。正确的判断方法是：(a>3) && (a<5)
* C语言规定：任何非0值都为“真”，只有0才为“假”。因此逻辑与也适用于数值。比如 5 && 4的结果是1，为“真”；-6 && 0的结果是0，为“假”

## 2.|| 逻辑或

1> 使用格式

“条件A || 条件B”

2> 运算结果

当条件A或条件B只要有一个成立时（也包括条件A和条件B都成立），结果就为1，也就是“真”；只有当条件A和条件B都不成立时，结果才为0，也就是“假”。

3> 运算过程

总是先判断条件A是否成立

* 如果条件A成立，就不会再去判断条件B是否成立：因为条件A已经成立了，不管条件B如何，“条件A || 条件B”的结果肯定是1，也就是“真”
* 如果条件A不成立，接着再判断条件B是否成立：如果条件B成立，“条件A || 条件B”的结果就为1，即“真”，如果条件B不成立，结果就为0，即“假”

4> 举例

逻辑或的结合方向是“自左至右”。比如表达式 (a<3) || (a>5)

* 若a的值是4：先判断a<3，不成立；再判断a>5，也不成立。因此结果为0
* 若a的值是2：先判断a<3，成立，停止判断。因此结果为1
* 因此，如果a的值在(-∞, 3)或者(5, +∞)范围内，结果就为1；否则，结果就为0

5> 注意

 C语言规定：任何非0值都为“真”，只有0才为“假”。因此逻辑或也适用于数值。比如 5 || 4的结果是1，为“真”；-6 || 0的结果是1，为“真”；0 || 0的结果是0，为“假”

## 3.! 逻辑非

1> 使用格式

“! 条件A”

2> 运算结果

其实就是对条件A进行取反：若条件A成立，结果就为0，即“假”；若条件A不成立，结果就为1，即“真”。也就是说：真的变假，假的变真。

3> 举例

逻辑非的结合方向是“自右至左”。比如表达式 ! (a>5)

* 若a的值是6：先判断a>5，成立，再取反之后的结果为0
* 若a的值是2：先判断a>3，不成立，再取反之后的结果为1

因此，如果a的值大于5，结果就为0；否则，结果就为1

4> 注意

* 可以多次连续使用逻辑非运算符：!(4>2)结果为0，是“假”，!!(4>2)结果为1，是“真”，!!!(4>2)结果为0，是“假”
* C语言规定：任何非0值都为“真”，只有0才为“假”。因此，对非0值进行逻辑非!运算的结果都是0，对0值进行逻辑非!运算的结果为1。!5、!6.7、!-9的结果都为0，!0的结果为1

## 4.优先级

* 逻辑运算符的优先级顺序为： 小括号() > 负号 - > ! > 算术运算符 > 关系运算符 > && > ||
* 表达式!(3>5) || (2<4) && (6<1) ：先计算 !(3>5)、(2<4)、(6<1)，结果为1，式子变为1 || 1 && 0，再计算1 && 0，式子变为1 || 0，最后的结果为1
* 表达式3+2<5||6>3 等价于 ((3+2) < 5) || (6>3)，结果为1
* 表达式4>3 && !-5>2 等价于 (4>3) &&  ((!(-5)) > 2) ，结果为0

# 三目运算符

N目运算符->三目运算符

* int a = 5?10:2;
* 获得a、b中的最大数
* 获得a、b、c中的最大数