C 语言位运算符：与、或、异或、取反、左移和右移

语言位运算符：与、或、异或、取反、左移和右移

位运算是指按二进制进行的运算。在系统软件中，常常需要处理二进制位的问题。C 语言提供了 6 个位操作运算符。这些运算符只能用于整型操作数，即只能用于带符号或无符号的 char,short,int 与 long 类型。

C 语言**提供的位运算符列表：**
运算符 含义 描述
& 按位与 如果两个相应的二进制位都为 1，则该位的结果值为 1，否则为 0
| 按位或 两个相应的二进制位中只要有一个为 1，该位的结果值为 1
^ 按位异或 若参加运算的两个二进制位值相同则为 0，否则为 1
~ 取反 ~是一元运算符，用来对一个二进制数按位取反，即将 0 变 1，将 1 变 0
<< 左移 用来将一个数的各二进制位全部左移 N 位，右补 0
>> 右移 将一个数的各二进制位右移 N 位，移到右端的低位被舍弃，对于无符号数，高位补 0

1、"按位与"运算符（&）

按位与是指：参加运算的两个数据，按二进制位进行"与"运算。如果两个相应的二进制位都为 1，则该位的结果值为 1；否则为 0。这里的 1 可以理解为逻辑中的 true,0 可以理解为逻辑中的 false。按位与其实与逻辑上"与"的运算规则一致。逻辑上的"与"，要求运算数全真，结果才为真。若，A=true,B=true,则 A∩B=true 例如：3&5 3 的二进制编码是 11(2)。（为了区分十进制和其他进制，本文规定，凡是非十进制的数据均在数据后面加上括号，括号中注明其进制，二进制则标记为 2）内存储存数据的基本单位是字节（Byte），一个字节由 8 个位（bit)所组成。位是用以描述电脑数据量的最小单位。二进制系统中，每个 0 或 1 就是一个位。将 11（2）补足成一个字节，则是 00000011（2）。5 的二进制编码是 101（2），将其补足成一个字节，则是 00000101（2）
按位与运算：
00000011(2)
&00000101(2)
00000001(2)
由此可知 3&5=1
c 语言代码：

```
#include <stdio.h>
main()
{
int a=3;
int b = 5;
printf("%d",a&b);
}
```

按位与的用途：
（1）清零

若想对一个存储单元清零，即使其全部二进制位为 0，只要找一个二进制数，其中各个位符合一下条件：

原来的数中为 1 的位，新数中相应位为 0。然后使二者进行&运算，即可达到清零目的。
例：原数为 43，即 00101011（2），另找一个数，设它为 148，即 10010100（2），将两者按位与运算：
00101011（2）
&10010100（2）
00000000（2）
c 语言源代码：

```
#include <stdio.h>
main()
{
int a=43;
int b = 148;
printf("%d",a&b);
}
```

（2）取一个数中某些指定位
若有一个整数 a(2byte),想要取其中的低字节，只需要将 a 与 8 个 1 按位与即可。
a 00101100 10101100
b 00000000 11111111
c 00000000 10101100
（3）保留指定位：
与一个数进行"按位与"运算，此数在该位取 1.
例如：有一数 84，即 01010100（2），想把其中从左边算起的第 3，4，5，7，8 位保留下来，运算如下：
01010100(2)
&00111011(2)
00010000(2)
即：a=84,b=59
c=a&b=16
c 语言源代码：

```
#include <stdio.h>
main()
{
int a=84;
int b = 59;
printf("%d",a&b);
}
```

2、"按位或"运算符（|）
两个相应的二进制位中只要有一个为 1，该位的结果值为 1。借用逻辑学中或运算的话来说就是，一真为真

。
例如：60（8）|17（8）,将八进制 60 与八进制 17 进行按位或运算。
00110000
|00001111
00111111
c 语言源代码：

```
#include <stdio.h>
main()
{
int a=060;
int b = 017;
printf("%d",a|b);
}
```

应用：按位或运算常用来对一个数据的某些位定值为 1。例如：如果想使一个数 a 的低 4 位改为 1，则只需要将 a 与 17（8）进行按位或运算即可。

3、交换两个值，不用临时变量

例如：a＝3，即 11（2）；b＝4，即 100（2）。

想将 a 和 b 的值互换，可以用以下赋值语句实现：

a＝a^b;

b＝b^a;

a＝a^b;

a＝011(2)

（^）b＝100(2)

a＝111(2)（a^b 的结果，a 已变成 7）

（^）b＝100(2)

b＝011(2)（b^a 的结果，b 已变成 3）

（^）a＝111(2)

a＝100（2）（a^b 的结果，a 已变成 4）

等效于以下两步：

① 执行前两个赋值语句：“a＝a^b；”和“b＝b^a；”相当于 b=b^(a^b)。

② 再执行第三个赋值语句： a ＝ a ∧ b 。由于 a 的值等于（ a ∧ b ），b 的值等于（ b ∧ a ∧ b ），

因此，相当于 a= a ∧ b ∧ b ∧ a ∧ b ，即 a 的值等于 a ∧ a ∧ b ∧ b ∧ b ，等于 b 。

很神奇吧！
c 语言源代码：

```
#include <stdio.h>
main()
{
int a=3;
int b = 4;
a=a^b;
b=b^a;
a=a^b;
printf("a=%d b=%d",a,b);
}
```

4、“取反”运算符（~）

他是一元运算符，用于求整数的二进制反码，即分别将操作数各二进制位上的 1 变为 0，0 变为 1。
例如：~77(8)
源代码：

```
#include <stdio.h>
main()
{
int a=077;
printf("%d",~a);
}
```

5、左移运算符（<<）

左移运算符是用来将一个数的各二进制位左移若干位，移动的位数由右操作数指定（右操作数必须是非负

值），其右边空出的位用 0 填补，高位左移溢出则舍弃该高位。
例如：将 a 的二进制数左移 2 位，右边空出的位补 0，左边溢出的位舍弃。若 a=15,即 00001111（2），左移 2

位得 00111100（2）。
源代码：

```
#include <stdio.h>
main()
```

```
{
int a=15;
printf("%d",a<<2);
}
```

左移 1 位相当于该数乘以 2，左移 2 位相当于该数乘以 2*2＝4,15＜＜2=60，即乘了４。但此结论只适用于该

数左移时被溢出舍弃的高位中不包含 1 的情况。

假设以一个字节（ 8 位）存一个整数，若 a 为无符号整型变量，则 a ＝64 时，左移一位时溢出的是 0

，而左移 2 位时，溢出的高位中包含 1。

6、右移运算符（>>）

右移运算符是用来将一个数的各二进制位右移若干位，移动的位数由右操作数指定（右操作数必须是非负

值），移到右端的低位被舍弃，对于无符号数，高位补 0。对于有符号数，某些机器将对左边空出的部分

用符号位填补（即"算术移位"），而另一些机器则对左边空出的部分用 0 填补（即"逻辑移位"）。注

意：对无符号数,右移时左边高位移入 0；对于有符号的值,如果原来符号位为 0(该数为正),则左边也是移

入 0。如果符号位原来为 1(即负数),则左边移入 0 还是 1,要取决于所用的计算机系统。有的系统移入 0,有的

系统移入 1。移入 0 的称为"逻辑移位",即简单移位；移入 1 的称为"算术移位"。

例： a 的值是八进制数 113755：

a:1001011111101101 （用二进制形式表示）

a>>1: 0100101111110110 (逻辑右移时)

a>>1: 1100101111110110 (算术右移时)

在有些系统中,a>>1 得八进制数 045766,而在另一些系统上可能得到的是 145766。Turbo C 和其他一些 C

编译采用的是算术右移,即对有符号数右移时,如果符号位原来为 1，左面移入高位的是 1。

源代码：

```
#include <stdio.h>
main()
{
int a=0113755;
printf("%d",a>>1);
}
```

7、位运算赋值运算符

位运算符与赋值运算符可以组成复合赋值运算符。

例如: &=, |=, >>=, <<=, ^=

例： a & = b 相当于 a = a & b

a << =2 相当于 a = a << 2