C 语言位运算

所谓位运算，就是对一个比特（Bit）位进行操作。在《二进制思想以及数据的存储》一节中讲到，比特（Bit）是一个电子元器件，8 个比特构成一个字节（Byte），它已经是粒度最小的可操作单元了。

C 语言提供了六种位运算符：

运算符	&	|	^	~	<<	>>
说明	按位与	按位或	按位异或	取反	左移	右移

按位与运算

一个比特（Bit）位只有 0 和1 两个取值，只有参与&运算的两个位都为 1 时，结果才为 1，否则为 0。例如 1&1 为1，0&0 为0，1&0 为0。

数值在内存中以二进制的形式存在，`9&5` 可写算式如下：

```
  00001001    (9 的二进制)
&00000101    (5 的二进制)
  00000001    (1 的二进制)
```

所以 `9&5=1` 。

> 严格来说，数值在内存中以补码形式存在，整数的补码与它的二进制形式相同，负数则不一样，不了解的读者可自行脑补。

按位与运算符&会对参与运算的两个数的所有二进制位进行&运算。

按位与运算通常用来对某些位清 0 或保留某些位。例如把 c 的高 16 位清 0 ，保留低 16 位，可作`a&65535`运算（65536 占用4 个字节，二进制数为00000000000000001111111111111111）。

【示例】位运算举例。

```
#include <stdio.h>
int main(){
    unsigned a=9;  //二进制数 00001001
    unsigned b=5;  //二进制数 00000101
    unsigned c=0XDE09A32B;  //十进制数 3725173547
    unsigned d=0X0000FFFF;  //十进制数 65535
    printf("a=%u, b=%u, a&b=%u\n", a, b, a&b);
    printf("c=%u, d=%u, c&d(%%d)=%u, c&d(%%X)=%X\n", c, d, c&d, c&d);
    return 0;
}
```

运行结果：

```
a=9, b=5, a&b=1

c=3725173547, d=65535, c&d(%d)=41771, c&d(%X)=A32 B
```

按位或运算

参与或运算|的两个二进制位有一个为 1 时，结果就为 1，两个都为 0 时结果才为 0。例如 1|1 为 1，0|0 为 0，1|0 为1。

9|5 可写算式如下：

```
  00001001    (9 的二进制)
 |00000101    (5 的二进制)
  00001101    (13 的二进制)
```

所以 9|5=13。

按位或运算可以用来将某些二进制位置1，而保留某些位。

【示例】或运算举例。

```
#include <stdio.h>
```

```
int main(){
unsigned a=9;  //二进制数 00001001
unsigned b=5;  //二进制数 00000101
unsigned c=0XDE09A30B;  //十进制数 3725173547
unsigned d=0XFFFF0000;  //十进制数 65535
    printf("a=%u, b=%u, a|b=%u\n", a, b, a|b);
    printf("c=%u, d=%u, c|d(%%d)=%u, c|d(%%X)=%X\n", c, d, c|d, c|d);
    return 0;
}
```

运行结果：

a=9, b=5, a|b=13

c=3725173515, d=4294901760, c|d(%d)=4294943499, c|d(%X)=FFFFA30 B

按位异或运算

参与异或运算^ 的两个二进制位不同时，结果为1，相同时结果为 0。也就是说，0^1 为 1，0^0 为 0，1^1 为 0。

9^5 可写成算式如下：

```
 00001001   (9 的二进制)
^00000101   (5 的二进制)
 00001100   (12 的二进制)
```

所以 9^5=12。

按位异或运算可以用来反转某些二进制位。

【示例】异或运算举例。

```
#include <stdio.h>
int main(){
    unsigned a=9;  //二进制数 00001001
    unsigned b=5;  //二进制数 00000101
    unsigned c=0X00FFFF00;  //十进制数 3725173547
```

```
    unsigned d=0XFFFF0000;  //十进制数 65535
    printf("a=%u, b=%u, a^b=%u\n", a, b, a^b);
    printf("c=%u, d=%u, c^d(%%d)=%u, c^d(%%X)=%X\n", c, d, c^d, c^d);
    return 0;
}
```

运行结果：

```
a=9, b=5, a^b=12
```

```
c=16776960, d=4294901760, c^d(%d)=4278255360, c^d(%X)=FF00FF00
```

取反运算

取反运算符~为单目运算符，右结合性，作用是对参与运算的数的各二进位按位取反。例如 ~1 为 0，~0 为1。

~9 的运算为：

```
~0000 0000 0000 1001
```

```
 1111 1111 1111 0110
```

这正是 -10 的补码表示，所以~9=-10。

左移运算

左移运算符<< 用来把操作数的各二进位全部左移若干位，高位丢弃，低位补 0。例如：

```
a=9;
a<<3;
```

<<左边是要移位的操作数，右边是要移动的位数。

上面的代码表示把 a 的各二进位向左移动 3 位。a=00001001(9 的二进制)，左移 3 位后为 01001000（十进制 72）。

右移运算

右移运算符>>用来把操作数的各二进位全部右移若干位，低位丢弃，高位补 0（或 1）。例如：

```
a=9;
a>>3;
```

表示把a 的各二进位向右移动3 位。a=00001001(9 的二进制)，右移 3 位后为00000001（十进制 1）。

需要注意的是，对于有符号数，在右移时，符号位将随同移动。当为正数时，最高位补0，而为负数时，符号位为1，最高位是补 0 或是补1 取决于编译器的规定。

【示例】位操作综合示例。

```
#include <stdio.h>
int main(){
    unsigned c=0X00FFFF00;  //十进制数 3725173547
    unsigned d=0XFFFF0000;  //十进制数 65535
    printf("c=%X, d=%X, c^d(%%X)=%X, c|d(%%X)=%X, c>>4=%X, c<<8=%X\n", c, d, c^d, c|d, c>>4, c<<8);
    return 0;
}
```

运行结果：

```
c=FFFF00,    d=FFFF0000,    c^d(%X)=FF00FF00,    c|d(%X)=FFFFFF00,    c>>4=FFFF0,
c<<8=FFFF0000
```