操作符详解

`1. 各种操作符的介绍。

2. 表达式求值`

@TOC

一、移位操作符

`<<` 左移操作符

`>>` 右移操作符

注：移位操作符的操作数只能是整数。

> 移位操作符移动的是存储在内存中的补码

1. 左移操作符 <<

移位规则：

左边抛弃、右边补0

```
# define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
//操作符详解

int main() {
    int a = 4;
    //00000000000000000000000000000100     正数的原码，反码，补码是相同的
    //左移操作
    int b = a << 1;  //把a向左移动一位
    // 0 00000000000000000000000000000100
    //最左边的一位将移出当前4个字节的内存单元，即少一位，那么末尾也丢失一位，进行补0
    //得到如下结果
    //00000000000000000000000000001000
    //则b为8
    printf("a = %d\nb = %d\n", a, b);
    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
a = 4
b = 8

D:\Microsoft Visual Studio\Code\C\Test_8_15\x64\Debug\Test_8_15.exe (进程 19436)已退出，代码为 0。
按任意键关闭此窗口. . .
```


考虑负数

```c
//考虑负数
int main() {
    int a = -4;
    //10000000000000000000000000000100  补码
    //11111111111111111111111111111011  反码
    //11111111111111111111111111111100  补码

    //补码左移一位,最左边1移除，最右边确实一位补0
    //11111111111111111111111111111000   b
    //
    //11111111111111111111111111110111   b的反码
    //10000000000000000000000000001000   b的原码（显然，b为-8）
    int b = a << 1;
    printf("a = %d\nb = %d\n", a, b);
}
```

```
Microsoft Visual Studio 调试控制台
a = -4
b = -8

D:\Microsoft Visual Studio\Code\C\Test_8_15\x64\Debug\Test_8_15.exe (进程 14756)已退出，代码为 0。
按任意键关闭此窗口. . ._
```


2. 右移操作符 >>

> 1.逻辑右移：右边丢弃，左边补0
> 2.算数右移：右边丢弃，左边补原符号位

大部分采用右移运算符

右移操作符,考虑-4

```c
int main() {
    int a = -4;
    //10000000000000000000000000000100  补码
    //11111111111111111111111111111011  反码
    //11111111111111111111111111111100  补码

    int b = a >> 1;
    //11111111111111111111111111111100
    //11111111111111111111111111111110  //右移一位的结果(补码)
    //11111111111111111111111111111101  得到反码
    //10000000000000000000000000000010  原码,结果为-2
    printf("a = %d\nb = %d\n", a, b);
    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
a = -4
b = -2

D:\Microsoft Visual Studio\Code\C\Test_8_15\x64\Debug\Test_8_15.exe (进程 19520)已退出，代码为 0。
按任意键关闭此窗口. . ._
```


显然验证4 >> 1会得到2，右移有除2的作用。

警告⚠ ：

对于移位运算符，不要移动负数位，这个是标准未定义的。

例如：

```
int num = 10;
num>>-1;//error
```

二、位操作符

二进制位的操作

&： 按位与 (都为1才为1，否则为0)

|： 按位或 (只要有1就为1，全0则得0)

^： 按位异或 (相同为0，相异为1)

注：他们的操作数必须是整数。

例如（按位与&）：

```
//位操作符(& | ^)
int main() {
    int a = 4;
    int b = -5;
    int c = a & b;  //二进制位与&
    //先得到a b的二进制补码
    //00000000000000000000000000000100  - a的补码
    //10000000000000000000000000000101  - b的原码
    //11111111111111111111111111111010  - b的反码
    //11111111111111111111111111111011  - b的补码
    //a,b的补码按每一位进行与操作(都为1得到1,否则为0)
    //00000000000000000000000000000000  - 结果为0
    printf("c = %d\n", c);
    return 0;
}
```

实例分析

1.不创建临时变量，实现两个数的交换

```
//1.不创建临时变量，实现两个数的交换
//先考虑一种简单的方法
int main() {
    int a = 5;
    int b = 8;

    a = a + b;
    b = a - b;
    a = a - b;
    printf("a = %d b = %d\n", a, b);
    return 0;
    //但是这样的做法有一个问题,就是如果a+b的值超过了int数值范围将不再适用
}
```

考虑异或特性

```
int main() {
    int a = 5;
    int b = 8;
    //原理如下
    //a^a=0
    //0^a=a
    a = a ^ b;
    b = a ^ b;  //等价于 b = a ^ b ^ b  可得到a
    a = a ^ b;  //相当于a^a^b
    printf("a = %d b = %d\n", a, b);
    return 0;
}
```

2. 写一个函数返回参数二进制中 1 的个数

比如： 15 0000 1111 4 个 1

```
int main() {
    int num = 1;
    int count = 0;
    for (int i = 1; i <= sizeof(num)*8; i++)
    {
        if (num & 1 == 1)
            count++;
        num = num >> 1;
    }
    printf("%d\n", count);
}
```

3. 输入两个整数，求两个整数二进制格式有多少个位不同

```
int main() {
    int a, b;
    int count = 0;
    scanf("%d %d", &a, &b);
    for (int i = 1; i <= 32; i++) {
        if (((a & 1) ^ (b & 1)) == 1)  //相异为1
            count++;
        a >>= 1;
        b >>= 1;
    }
    printf("%d\n", count);
    return 0;
}
```