本小节内容

位运算符
异或运算符实例解析

位运算符

位运算符<<、>>、~、|、^、&依次是左移、右移、按位取反、按位或、按位异或、按位与。

位运算符只能用于对整型数据进行操作。

左移：高位丢弃，低位补 0，相当于乘以 2。工作中很多时候申请内存时会用左移，例如要申请 1GB 大小的空间，可以使用 malloc(1<<30)。

右移：低位丢弃，正数的高位补 0（无符号数我们认为是正数），负数的高位补 1，相当于除以 2。移位比乘法和除法的效率要高，负数右移，对偶数来说是除以 2，但对奇数来说是先减 1 后除以 2。例如，–8>>1，得到的是–4，但–7>>1 得到的并不是–3 而是–4。另外，对于–1 来说，无论右移多少位，值永远为–1。

异或：相同的数进行异或时，结果为 0，任何数和 0 异或的结果是其本身。

按位取反：数位上的数是 1 变为 0，0 变为 1。

按位与和按位或：用两个数的每一位进行与和或。

请看下面实例：

```
#include <stdio.h>

int main()
{
	short i = 5;
	short j;
	j=i << 1;//一个变量移动以后自身不会变化
	printf("j=%d\n", j);//左移是乘 2，结果为 10
	j = i >> 1;
	printf("j=%d\n", j);//右移是除 2，结果是 2
	i = 0x8011;
	unsigned short s = 0x8011;
	unsigned short r = 0;
	j = i >> 1;//对 i 右移
	r = s >> 1;//对 s 右移
	printf("j=%d,r=%u\n", j, r);//结果是不一样的
	//接下来来看  按位与，按位或，按位异或，按位取反
	i = 5, j = 7;
	printf("i & j=%d\n", i & j);
	printf("i | j=%d\n", i | j);
	printf("i ^ j=%d\n", i ^ j);
```

```
    printf("~i=%d\n", ~i);
    return 0;

}
```

异或运算符实例解析

异或运算符有两个特性，一是任何数和零异或得到的是自身，两个相等的数异或得到的是零，通过这两个特性，我们可以完成下面的题目，在一堆数中找出出现 1 次的那个数。

```
#include <stdio.h>

int main()
{
    int i;
    int arr[5] = { 8 ,5, 3, 5, 8 };
    int result = 0;
    for (i = 0; i < 5; i++)
    {
        result ^= arr[i];
    }
    printf("%d\n", result);//输出为 3
    return 0;
}
```