

计算机程序设计

Computer Programming



位运算



主讲：吴锋

目录

CONTENTS

位运算和位运算符

结构体中的位

位运算的例子



◎ 位运算的概念

- 很多算法是按二进制位进行运算的
- 位运算速度快，效率高
- 位运算的运算对象是二进制的位
- 只能对整型数据(包括字符型)进行位运算
- 负数以补码形式参与运算



◎ 位运算符

- 注意位运算符与逻辑运算区别

运算符	运算	举例	优先级从高到低 (逻辑非运算符)
~	按位取反	~flag	(算术运算符)
<<	左移	a << 2	(关系运算符)
>>	右移	b >> 3	(赋值运算符)
&	按位与	flag & 0x37	
^	按位异或	flag ^ 0xC4	
	按位或	flag 0x5A	

按位与 (Bitwise AND) &

• 运算规则

- $0 \& 0 = 0;$
- $0 \& 1 = 0;$
- $1 \& 0 = 0;$
- $1 \& 1 = 1;$

	1010, 1101	(0xAD)
&	0110, 1001	(0x69)
<hr/>		
	0010, 1001	(0x29)

• 特殊用法

- 特定位清零
- 保留其它位

	xxxx, xxxx	
&	0110, 0010	(0x62)
<hr/>		
	0xx0, 00x0	

按位或 (Bitwise Inclusive OR)

- 运算规则

- $0 \mid 0 = 0;$
- $0 \mid 1 = 1;$
- $1 \mid 0 = 1;$
- $1 \mid 1 = 1;$

	1010, 1101	(0xAD)
	0110, 1001	(0x69)
<hr/>		
	1110, 1101	(0xED)

- 特殊用法

- 特定位置一
- 保留其它位

	xxxx, xxxx	
	0110, 0010	(0x62)
<hr/>		
	x11x, xx1x	

◎ 按位异或 (Bitwise Exclusive OR) \wedge

• 运算规则

- $0 \wedge 0 = 0$;
- $0 \wedge 1 = 1$;
- $1 \wedge 0 = 1$;
- $1 \wedge 1 = 0$;

	1010, 1101	(0xAD)
\wedge	0110, 1001	(0x69)
<hr style="border: 0.5px solid black;"/>		
	1100, 0100	(0xC4)

• 特殊用法

- 特定位取反
- 保留其它位

	xxxx, xxxx	
\wedge	0110, 0010	(0x62)
<hr style="border: 0.5px solid black;"/>		
	x \bar{x} x \bar{x} , x \bar{x} x \bar{x}	

◎ 按位取反 (Bitwise NOT) ~

- 运算规则

- $\sim 0 = 1;$

- $\sim 1 = 0;$

~	0110,1001	(0x69)
<hr/>		
	1001,0110	(0x96)

◎ 左移 (Left Shift) <<

- 运算规则

- $i \ll n$
- 把*i*各位全部向左移动*n*位，**不会改变*i*的值**
- 最左端的*n*位被移出丢弃
- 最右端的*n*位用0补齐

$$5 \ll 3 = 40$$

$$00000101 \ (0x05) \ll 3 \rightarrow 00101000 \ (0x28)$$

- 用法

- 若没有溢出，则左移*n*位相当于乘上 2^n
- 运算速度比真正的乘法和幂运算快得多



◎ 右移 (Right Shift) >>

- 运算规则

- $i \gg n$
 - 把 i 各位全部向右移动 n 位, 不会改变 i 的值
 - 最右端的 n 位被移出丢弃
 - 最左端的 n 位用 0 补齐(逻辑右移)
 - 或最左端的 n 位用符号位补齐(算术右移)
- 由编译系统的实现者决定。
为了可移植性, 最好仅对无符号数进行移位运算

$$5 \gg 2 = 1$$

$$00000101 \text{ (0x05)} \gg 2 \rightarrow 00000001 \text{ (0x01)}$$

- 用法

- 右移 n 位相当于除以 2^n , 并舍去小数部分
- 运算速度比真正的除法和幂运算快得多

◎ 位运算

- 按位运算时，两个操作数长度应相等，
- 否则先扩展，再运算
 - 两个操作数右端对齐
 - 短的数据左端用符号位补齐
 - 正数或无符号数左端用0补满
 - 负数左端用1补满
- 位运算符都不会改变参与运算变量的值
- 复合赋值运算符（改变参与运算变量的值）
 $\&=$ ， $\wedge=$ ， $|=$ ， $\sim=$ ， $\ll=$ ， $\gg=$



◎ 位运算

- 例：如何判断一个int型变量n的第7位（从右往左，从0开始数）是否是1？
- 答：只需看 $n \& 0x80$ 的值是否等于0x80即可。

$$\begin{array}{r}
 \text{XXXX XXXX XXXX XXXX XXXX XXXX } \textcolor{red}{\text{X}}\text{XXX XXXX } (n) \\
 \& \quad 0000 0000 0000 0000 0000 0000 \textcolor{red}{1}000 0000 (0x80) \\
 \hline
 0000 0000 0000 0000 0000 0000 \textcolor{red}{x}000 0000
 \end{array}$$

◎ 位运算

- 例：将16进制短整数按二进制打印输出

- 输入：F1E2
- 输出：1111000111100010
- 输入：13A5
- 输出：0001001110100101

```
#include <stdio.h>
void main()
{
    int i;
    short a;
    scanf("%X", &a);
    for (i=15;i>=0;i--)
        printf("%1d", a&1<<i ? 1 : 0);
}
```

◎ 位运算

- 异或运算的特点是：
 - 如果 $a \oplus b = c$ ，那么就有 $c \oplus b = a$ 以及 $c \oplus a = b$
- 此规律可以用来进行最简单的加密和解密以及信息校验
 - 加密：明文 \oplus 密钥 = 密文；解密：密文 \oplus 密钥 = 明文
 - 校验：信息 (a b c d) + 校验码 ($a \oplus b \oplus c \oplus d$)
- 还可以用来交换两位数a和b
 - $a = a \oplus b$
 - $b = a \oplus b$ ，即： $(a \oplus b) \oplus b = a$
 - $a = a \oplus b$ ，即： $(a \oplus b) \oplus a = b$



- 例：在一个整数数组中，仅存在一个不重复的数字，其余数字均出现两次（或偶数次），找出不重复数字。

```
for (int i = 0; i < N; ++i) {  
    int c = 0;  
    for (int j = 0; j < N; ++j)  
        if (i != j && a[i] == a[j])  
            ++c;  
    if (c % 2 != 0)  
        return a[i];  
}
```

双重循环的解法 $O(N^2)$

```
int r = 0;  
for (int i = 0; i < N; ++i)  
    r = r ^ a[i];  
return r;
```

按位异或的解法 $O(N)$

$$0 \wedge a = a$$

$$a \wedge a = 0$$

$$a \wedge a \wedge a = a$$

$$a \wedge a \wedge a \wedge a = 0$$

$$a \wedge b \wedge a \wedge a \wedge b = a$$

◎ 位运算

`int` `n1` = 15 : 0000 0000 0000 0000 0000 0000 0000 1111

`short` `n2` = 15 : 0000 0000 0000 1111

`unsigned short` `n3` = 15 : 0000 0000 0000 1111

`unsigned char` `c` = 15 : 0000 1111

`n1` <<= 15, (变成 78000) : 0000 0000 0000 0111 1000 0000 0000 0000

`n2` <<= 15, (变成 -32768) : 1000 0000 0000 0000

`n3` <<= 15, (变成 32768) : 1000 0000 0000 0000

`c` <<= 6 , (变成 0xc0) : 1100 0000

`int` `n4` = `c` << 4 这个表达式是先将 `c` 转换成整型:

0000 0000 0000 0000 0000 0000 1100 0000

然后再左移, 即: `(int)(c<<4)=3072`



◎ 位运算

`int n1 = 15` : 0000 0000 0000 0000 0000 0000 0000 1111

`short n2 = -15` : 1111 1111 1111 0001

`unsigned short n3 = 0xffe0` : 1111 1111 1110 0000

`unsigned char c = 15` : 0000 1111

`n1 >>= 2, 变成3` : 0000 0000 0000 0000 0000 0000 0000 0011

`n2 >>= 3, 变成-2` : 1111 1111 1111 1110

`n3 >>= 4, 变成 0xffe` : 0000 1111 1111 1110

`c >>= 3, 变成1` : 0000 0001

◎ 位运算

- 例：有两个个int型变量a和n ($0 \leq n \leq 31$)，求a的第n位（从右往左，从0开始数）的值？
- 答：(a >> n) & 1 或 (a & (1 << n)) >> n。

XXXX XXXX XXXX XXXX XXXX XXXX **X**XXX XXXX (a)

0000 000**x** XXXX XXXX XXXX XXXX XXXX XXX**X** (>> 7)

& 0000 0000 0000 0000 0000 0000 0000 000**1** (1)

0000 0000 0000 0000 0000 0000 0000 000**x**

◎ 结构体中的位

- 位段是以位为单位定义长度的结构体类型成员
- 定义形式:

```
struct 位域结构名 {  
    位段成员列表;  
};
```

位段成员的说明形式为:

类型说明符 位段名:位段长度;

- 注意:
 - 一个位段分配在同一个存储单元之中, 不能跨单元
 - 可以用 unsigned int:0; 表示从下一个存储单元开始存放
 - 可以用未命名的位段来填充或调整位置, 例如 unsigned int:3;

```
struct packed_data {  
    unsigned int a:2;  
    unsigned int b:6;  
    unsigned int c:4;  
    unsigned int d:4;  
    unsigned int :0;  
    int i;  
};
```

位段成员

普通结构体成员

◎ 结构体中的位（例）

```
struct Date {
    unsigned int d;
    unsigned int m;
    unsigned int y;
};
// 输出: 12 bytes, 每个存储单元 4 bytes
printf("%lu bytes\n", sizeof(struct date));
```

```
struct Date {
    unsigned int d : 5; // 0~31天, 5 bits 就够了
    unsigned int m : 4; // 0~12月, 4 bits 就够了
    unsigned int y;
};
// 输出: 8 bytes, 共两个存储单元
printf("%lu bytes\n", sizeof(struct date));
```

```
struct date {
    unsigned int d : 5;
    unsigned int : 0; // 从下一个存储单元开始
    unsigned int m : 4;
    unsigned int y;
};
// 输出: 12 bytes, 共三个存储单元
printf("%lu bytes\n", sizeof(struct date));
```


◎ 结构体中的位

- 使用中应注意成员所占的位及其由长度限定的存取值域
 - 溢出了会怎样？与编译器的实现相关。
- 不能对位段应用取地址运算
 - 地址的最小运算单位是字节 (byte)
- 不能用指针指向位段
 - struct S { unsigned int x : 5; }; S s;
 - printf("%p", &s.x); // 错误
- 位段数组不被允许
 - struct S { unsigned int x[10] : 5; }; // 错误

```
#include <stdio.h>
struct test {
    unsigned int x : 2;
    unsigned int y : 2;
    unsigned int z : 2;
};
void main() {
    struct test t;
    t.x = 5;
    printf("%d %d", t.x, t.y);
}
// 可能的输出: 1 0
// 溢出的位被丢弃
// 未影响另一成员
```



◎ 位运算举例

- 例：求char型数据二进制表示中1的个数
- 方法1：模拟进制转换，反复除2，累加余数

```
int count1(unsigned char v)
{
    int num = 0;
    while (v) {
        if (v%2 == 1) num++;
        v /= 2;
    }
    return(num);
}
```

普通
程序员

十进制到二进制的转换



◎ 位运算举例

- 例：求char型数据二进制表示中1的个数
- 方法2：将方法1中运算改为位运算

```
int count2(unsigned char v)
{
    int num=0;
    while (v) {
        num += v & 0x01;
        v >>= 1;
    }
    return(num);
}
```

高级
程序员

位运算快于算术运算



◎ 位运算举例

- 例：求char型数据二进制表示中1的个数
- 方法3：将循环次数降为1的个数

思路：设法使得每次循环都能减少一个1

```
int count3(unsigned char v) {  
    int num=0;  
    while (v) {  
        v &= (v-1);  
        num++;  
    }  
    return(num);  
}
```

算法
工程师

减1会使得最后一个1变为0，与之后1就少了一个



◎ 位运算举例

- 例：求char型数据二进制表示中1的个数
- 方法4：不用循环用判断分支，“手动判断”

```
int count4(unsigned char v)
{
    int num=0;
    switch (v) {
        case 0x00: num=0; break;
        .....
    }
    return(num);
}
```

码农密集型
企业

◎ 位运算举例

- 例：求char型数据二进制表示中1的个数
- 方法5：预存结果，“以空间换时间”

```
int count5(unsigned char v)
{
    static int countTable[256]={0, 1, ...};
    return countTable[v];
}
```

人工智能
企业