C 语言

第 15 讲、位操作

张晓平

武汉大学数学与统计学院

2017 年 6 月 7 日

1. C 的位运算符

1. C 的位运算符

C 提供位的逻辑运算符和移位运算符。

1.1 位逻辑运算符

C 语言提供了六种位逻辑运算符：

- 位与
- 位或
- 位异或
- 取反

这些运算符只能用于整型数据，即只能用于带符号或无符号的 `char`，`short`，`int` 与 `long` 类型。将这些运算符称为位 (bitwise) 的原因是它们对每个位进行操作，而不影响左右两侧的位。

注　请不要将这些运算符与常规的逻辑运算符相混淆（&&，||，!），常规的逻辑运算符对整个值进行操作。

一、二进制反码或按位取反：~

一元运算符~ 将每个 1 变为 0，将每个 0 变为 1。

例

```
~   00111011
----------------------
    11000100
```

设 val 是一个 unsigned char，已赋值为 2。因 2 = 00000010(2)，故~val 的值为 11111101，即 253。

设 val 是一个 unsigned char，已赋值为 2。因 2 = 00000010(2)，故~val 的值为 11111101，即 253。

注　该运算符不改变 val 的值。

若想将 val 的值变为~val，可以这样做：

```
val = ~val;
```

二、位与：&

& 运算符通过对两个操作数逐位进行比较产生一个新值。对于每个位，只有两个操作数的对应位都为 1，结果才为 1。

```
    00000011
&   00000101
-------------
    00000001
```

注 C 也有一个位与-赋值运算符：`&=`。

以下两条语句等效：

```
val &= 0377;
val = val & 0377;
```

三、位或：|

| 运算符通过对两个操作数逐位进行比较产生一个新值。对于每个位，如果任意操作数中对应的位为 1，那么结果位就为 1。

```
    10010011
|   00111101
--------------
    10111111
```

注 C 也有一个位或-赋值运算符：|=。

以下两条语句等效：

```
val |= 0377;
val = val | 0377;
```

四、位异或：^

^运算符通过对两个操作数逐位进行比较产生一个新值。对于每个位，如果任意操作数中对应的位有一个为 1，但不都为 1，那么结果位就为 1。

```
    10010011
^   00111101
--------------
    10101110
```

注 C 也有一个位异或-赋值运算符：^=。

以下两条语句等效：

```
val ^= 0377;
val = val ^ 0377;
```

1.2 用法：掩码

位与运算符通常跟掩码一起使用。掩码是某些位设为开（1）而某些位设为关（0）的位组合。

例　设已经定义符号常量 MASK 为 2，即二进制的 00000010，只有位 1 是非零。则以下语句

```
flags = falgs & MASK;
```

将导致 flags 除位 1 外的所有位都设置为 0。因掩码中的 0 覆盖了 flags 中相应的位，故该过程称为“使用掩码”。

以此类推，可将掩码中的 `0` 看做不透明，将 `1` 看做透明。表达式 `flags & MASK` 就好像使用掩码覆盖 `flags`：`flags` 中的位只有在 `MASK` 中的对应位是 `1` 时才可见。

可使用“位与-赋值”运算符来简化代码，如

```
flag &= MASK;
```

可使用“位与-赋值”运算符来简化代码，如

```
flag &= MASK;
```

一种常见的 C 用法如下

```
ch &= 0xff;
```

因 0xff 的二进制形式为 11111111，该掩码留下 ch 的低 8 位，将其余位设为 0。

1.3 用法：打开位

有时，可能需要打开一个值中特定的位，同事保持其他位不变。例如，一台 PC 通过将知发送到端口来控制硬件。如要打开扬声器，可能需要打开某一位，同时保持其他位不变。可以使用“位或”运算符来实现。

例 设 MASK 为 00000010，以下语句

```
flags = flags | MASK;
```

将 flags 中的位 1 设为 1，并保留其他所有位不变。

例　设 MASK 为 00000010，以下语句

```
flags = flags | MASK;
```

将 flags 中的位 1 设为 1，并保留其他所有位不变。

作为缩写，可使用位或-赋值运算符：

```
flags |= MASK;
```

1.4 用法：关闭位

关闭特定位与打开特定位是同样有用的。假设想关闭 `flags` 中的位 `1`。设 `MASK` 为 `00000010`，则以下语句

```
flags = flags & ~MASK;
```

将关闭位 `1`，而其他所有位不变。其缩写形式为

```
flags &= ~MASK;
```

1.5 用法：转置位

转置一个位表示如果该位打开，则关闭该位；如果该位关闭，则打开该位。可使用“位异或”运算符来转置一个位。

其思想为：设 b 是一个位（1 或 0），则

- 若 b 为 1，则 1^b 为 0；
- 若 b 为 0，则 1^b 为 1；
- 无论 b 是 0 还是 1，0^b 总是为 b。

因此，若使用^将一个值与掩码组合，则该值中对应掩码位为 1 的位被转置，对应掩码位为 0 的位不改变。

例 要转置 `flag` 中的位 `1`，可使用以下任一语句：

```
flag = flag ^ MASK;
flag ^= MASK;
```

1.6 用法：查看某一位的值

问题　我们已经知道了改变一位的值的方法，那么如何来查看某一位的值呢？

例 查看 `flag` 的位 `1` 的值。

例 查看 `flag` 的位 `1` 的值。

不能简单的比较 `flag` 与 `MASK (00000010)`：

```
if (flag == MASK)
  puts("Wow!");
```

例 查看 flag 的位 1 的值。

不能简单的比较 flag 与 MASK (00000010)：

```
if (flag == MASK)
  puts("Wow!");
```

必须屏蔽 flag 中的其他位，以便只把 flag 中的位 1 和 MASK 做比较：

```
if ( (flag & MASK) == MASK)
  puts("Wow!");
```

因位运算符的优先级低于 ==，故需在 flag & MASK 的两侧加圆括号。

1.7　移位操作符

移位运算符将位向左或向右移。

一、左移：«

左移运算符 « 将其左侧操作数的值的每位向左移动，移动的位数由其右侧操作数指定。空出的位用 0 填充，并且丢弃移出左侧操作数末端的位。

```
10001010 << 2
-------------
00101000
```

该操作产生一个新值，但不改变其操作数。如，设 a 为 1，则 a « 2 为 4，但 a 仍为 1。

可使用“左移-赋值”运算符（«=）来实际改变一个变量的值。

```
int a = 1;
int b;
b = a << 2; // assign 4 to b
a <<= 2;    // change a to 4
```

二、右移：»

左移运算符 » 将其左侧操作数的值的每位向右移动，移动的位数由其右侧操作数指定。丢弃移出左侧操作数右端的位。对于 `unsigned` 类型，使用 `0` 填充左端空出的位；对于有符号类型，结果依赖于机器：空出的位可能用 `0` 填充，或使用符号（最左端的）位的副本来填充。

对有符号值，

```
10001010 >> 2  // signed
--------------
00100010       // some os

10001010 >> 2  // signed
--------------
11100010       // the other os
```

对无符号值，

```
10001010 >> 2  // signed
--------------
00100010       // any os
```

可使用“右移-赋值”运算符（»=）来实际改变一个变量的值。

```
int a = 16;
int b;
b = a >> 2; // b = 2, a = 16
a >>= 2;    // a = 2
```

三、用法：移位运算符

移位运算符能够提供快捷、高效的（依赖于硬件）对 2 的幂的乘法和除法。

number « n	number 乘以 2 的 n 次幂
number » n	如果 number 非负，则用 number 除以 2 的 n 次幂

这些移位运算符类似于在十进制中移动小数点来乘以或除以 10。

移位运算符也用于从较大的单位中提取多组比特位。

例　假设使用一个 `unsigned long` 值代表颜色值，其中低位字节存放红色亮度，下一字节存放绿色亮度，第三个字节存放蓝色亮度。如何将每种颜色的亮度存储在各自的 `unsigned char` 变量中。

```c
#define MASK 0xff
unsigned long color = 0x002a162f;
unsigned char blue, green, red;
red = color & MASK;
green = (color >> 8)  & MASK;
blue  = (color >> 16) & MASK;
```

这段代码使用右移运算符将 8 位颜色值移动到低字节，然后使用掩码技术将低位字节赋给相应的变量。

1.8 编程实例

例　使用移位运算符实现一个十进制整数的二进制转码。要求编写一个 `itobs()` 函数，它接受两个参数，第一个是该整数，另一个是字符串地址。

```c
// binbit.c:
#include <stdio.h>
char * itobs(int n, char * ps);
void show_bstr(const char * str);

int main(void)
{
  char bin_str[8 * sizeof(int) + 1];
  int number;

  puts("Enter integers and see them in binary.");
  puts("Non-numeric input terminates program.");
  while(scanf("%d", &number) == 1) {
    itobs(number, bin_str);
    printf("%d is ", number);
```

```
16      show_bstr(bin_str);
17      putchar('\n');
18    }
19    puts("Bye!");
20
21    return 0;
22  }
23
24  char * itobs(int n, char * ps)
25  {
26    int i;
27    static int size = 8 * sizeof(int);
28
29    for (i = size-1; i >= 0; i--, n >>= 1)
30      ps[i] = (01 & n) + '0';
31    ps[size] = '\0';
```

```
32
33    return ps;
34 }
35
36 void show_bstr(const char * str)
37 {
38    int i = 0;
39    while (str[i]) {
40       putchar(str[i]);
41       if (++i % 4 == 0 && str[i])
42          putchar(' ');
43    }
44 }
```

```
Enter integers and see them in binary.
Non-numeric input terminates program.
7 [enter]
7 is 0000 0000 0000 0000 0000 0000 0000 0111
2017 [enter]
2017 is 0000 0000 0000 0000 0000 0111 1110 0001
-1 [enter]
-1 is 1111 1111 1111 1111 1111 1111 1111 1111
-2017 [enter]
-2017 is 1111 1111 1111 1111 1111 1000 0001 1111
q [enter]
Bye!
```

例　编写一个函数，反转一个值中的最后 n 位，参数为 n 和要反转的值。

取反运算符~可以反转位，但它反转一个值中的所有位，而不是选定的少数位。然而，异或运算符^可以用与转置单个位。

取反运算符~可以反转位，但它反转一个值中的所有位，而不是选定的少数位。然而，异或运算符^可以用与转置单个位。

创建一个掩码，它的最后 n 位设为 1，其他位设为 0。然后对该掩码和一个值使用异或运算符^就可以反转这个值的最后 n 位，同事保留该值的其他位不变。

```
int invert_end(int num, int bits)
{
  int mask = 0;
  int bitva}l = 1;
  while (bits-- > 0) {
    mask |= bitval;
    bitval <<= 1;
  }
  return num ^ mask;
}
```

该代码中，while 循环创建该掩码。最初，mask 的所有位都被设为 0。第一次执行该循环将位 0 设为 1，然后将 bitval 增加到 2；也就是将 bitval 的位 0 设为 0，位 1 设为 1。下次循环时，将 mask 的位 1 设为 1，以此类推。

```c
/* invert4.c -- using bit operations to display
binary */
#include <stdio.h>
char * itobs(int, char *);
void show_bstr(const char *);
int invert_end(int num, int bits);
int main(void)
{
  char bin_str[8 * sizeof(int) + 1];
  int number;
  puts("Enter integers and see them in binary.");
  puts("Non-numeric input terminates program.");
  while (scanf("%d", &number) == 1) {
    itobs(number,bin_str);
    printf("%d is\n", number);
```

```
15      show_bstr(bin_str);
16      putchar('\n');
17      number = invert_end(number, 4);
18      printf("Inverting the last 4 bits gives\n");
19      show_bstr(itobs(number,bin_str));
20      putchar('\n');
21    }
22    puts("Bye!");
23    return 0;
24  }
25
26  char * itobs(int n, char * ps)
27  {
28    int i;
29    static int size = 8 * sizeof(int);
30
```

```
31    for (i = size - 1; i >= 0; i--, n >>= 1)
32      ps[i] = (01 & n) + '0';
33    ps[size] = '\0';
34    return ps;
35  }
36
37  /* show binary string in blocks of 4 */
38  void show_bstr(const char * str)
39  {
40    int i = 0;
41    while (str[i]) { /* not the null character */
42      putchar(str[i]);
43      if(++i % 4 == 0 && str[i])
44        putchar(' ');
45    }
46  }
```

```
47
48 int invert_end(int num, int bits)
49 {
50   int mask = 0;
51   int bitval = 1;
52   while (bits-- > 0) {
53     mask |= bitval;
54     bitval <<= 1;
55   }
56   return num ^ mask;
57 }
```

```
Enter integers and see them in binary.
Non-numeric input terminates program.
7
7 is
0000 0000 0000 0000 0000 0000 0000 0111
Inverting the last 4 bits gives
0000 0000 0000 0000 0000 0000 0000 1000
1234
1234 is
0000 0000 0000 0000 0000 0100 1101 0010
Inverting the last 4 bits gives
0000 0000 0000 0000 0000 0100 1101 1101
q
Bye!
```