第四章 C语言的补充概念

模块4.2：位运算

4.2 位运算

4.2.1 概述

- 字节和位

  字节：byte，计算机中数据表示的基本单位

  位 ：bit， 计算机中数据表示的最小单位

  1 byte = 8 bits

- 位运算

  以bit为单位进行数据的运算

4.2 位运算

4.2.1 概述

- 位运算的基本方法
  - 按位进行（只有0、1）
  - 要求运算数据长度相等，若不等，则右对齐，按最高位补齐左边

```
char a=0x37;          0000 0000 0011 0111
short b=0x1234;       0001 0010 0011 0100
char a=0xA7;          1111 1111 1010 0111
short b=0x8341;       1000 0011 0100 0001
```

  - 数在计算机内是用补码表示的

4.2 位运算

4.2.2 常用的位运算

运算符	功能	用法
~	位求反	~expr
<<	左移	expr1 << expr2
>>	右移	expr1 >> expr2
&	位与	expr & expr
^	位异或	expr ^ expr
|	位或	expr | expr

4.2 位运算

- 移位运算符
  - 左移运算规则：左移数据，右补0
  - 右移运算规则：右移数据，左补0（逻辑右移）

    右移数据，左补符号位（算术右移）- C/C++的位运算

例： unsigned char bits=0233;

1	0	0	1	1	0	1	1

bits << 8; //bits提升成int类型，然后向左移动8位

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0

bits << 31; //向左移动31位，左边超出边界的位丢弃

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

bits >> 3; //向右移动3位，最右边的3位丢弃

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        unsigned char bits = 0233;
        cout << "bits<<8=0x" << hex << (bits << 8) << " "
                             << dec << (bits << 8) << endl;
        cout << "bits<<31=0x"<< hex << (bits << 31)<< " "
                             << dec << (bits << 31)<< endl;
        cout << "bits>>3=0x" << hex << (bits >> 3) << " "
                             << dec << (bits >> 3) << endl;

        return 0;
}
```

```
bits<<8=0x9b00 39680
bits<<31=0x80000000 -2147483648
bits>>3=0x13 19
```

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    unsigned char bits = 0233;
    cout << "bits<<8=0x" << hex << (int)(char)(bits << 8) << " "
                         << dec << (int)(char)(bits << 8) << endl;
    cout << "bits<<31=0x"<< hex << (int)(char)(bits << 31) << " "
                         << dec << (int)(char)(bits << 31) << endl;
    cout << "bits>>3=0x" << hex << (int)(char)(bits >> 3) << " "
                         << dec << (int)(char)(bits >> 3) << endl;
    return 0;
}
```

```
bits<<8=0x0 0
bits<<31=0x0 0
bits>>3=0x13 19
```

//蓝框部分不必要，可省略

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    char bits = 0233;  //提升为int类型时，填充的是符号位
    cout << "bits<<8=0x" << hex << (bits << 8) << " "
                         << dec << (bits << 8) << endl;
    cout << "bits<<31=0x"<< hex << (bits << 31) << " "
                         << dec << (bits << 31) << endl;
    cout << "bits>>3=0x" << hex << (bits >> 3) << " "
                         << dec << (bits >> 3) << endl;
    return 0;
}
```

```
bits<<8=0xffff9b00 -25856
bits<<31=0x80000000 -2147483648
bits>>3=0xfffffff3 -13
```

4.2 位运算

- 移位运算符
  - 左移：在不溢出(1不被舍去)的情况下，左移n位等于乘2的n次方(当做无符号数理解)
  - 右移：在不溢出(1不被舍去)的情况下，右移n位等于除2的n次方(当作有符号数理解)

4.2 位运算

- 移位运算符
  - 重载版本--IO运算符
  - 满足左结合律：

```
cout << 42 + 10; //正确：+优先级高，输出求和结果

cout << (10 < 42); //正确：输出1

cout << 10 < 42;  //错误：把数字10写到cout，然后将结果
                  （即cout）与42进行比较
```

4.2 位运算

- 位求反运算符
  - 运算规则：0/1互反

例： unsigned char bits = 0227;

1	0	0	1	0	1	1	1

~bits

//提升成int类型，原来位保持不变，高位添加0：

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1

//逐位求反：

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	0	0	0

4.2 位运算

- 位求反运算符
  - 运算对象：整数类型

例：求表达式的值 ~'q' << 6

//按运算符优先级，先对q按位取反：'q'转换为整数：

0	1	1	1	0	0	0	1

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1

//逐位求反：

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0

//移位操作：

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0

4.2 位运算

- 位与、位或、位异或运算符
  - 位与运算规则：遇0得0
  - 位或运算规则：遇1得1
  - 位异或运算规则：相同为0，不同为1

例：

```
unsigned char b1 = 0145;
unsigned char b2 = 0257;
b1 & b2
b1 | b2
b1 ^ b2
```

	0	1	1	0	0	1	0	1
	1	0	1	0	1	1	1	1
24个高阶位都是0	0	0	1	0	0	1	0	1
24个高阶位都是0	1	1	1	0	1	1	1	1
24个高阶位都是0	1	1	0	0	1	0	1	0

4.2 位运算

- 位与、位或、位异或运算符
  - 位运算符：　&　　|　　^
  - 逻辑运算符：&&　　||　　!

例：　unsigned long ul1 = 3, ul2 = 7;

```
(1) ul1 & ul2;          //3，占4个字节
(2) ul1 | ul2;          //7，占4个字节
(3) ul1 && ul2;         //true，占1个字节
(4) ul1 || ul2;         //true，占1个字节
```

4.2 位运算

- 复合位运算符
  - &=　　|=　　^=　　<<=　　>>=

```
...
int main()
{
    char a=0x18;
    int i;
    for(i=1; i<=6; i++) {
        a = a>>1;     //a >>= 1;
        cout << "a>>" << i << "=0x"  << hex << int(a) << " "
                                     << dec << int(a) << endl;
    }
}
```

```
a>>1=0xc 12
a>>2=0x6 6
a>>3=0x3 3
a>>4=0x1 1
a>>5=0x0 0
a>>6=0x0 0
```

4.2 位运算

4.2.3 位运算的应用

- 与(&)的应用
  - 清零

```
char a1=0xb6;
cout << "char a=" << hex << (int)a1 << endl
     << " a&0x0=" << dec << (a1&0x0) << endl;
```

例：char a1=0xb6; 现要求将该数清零，则：

```
  1011 0110
& 0?00 ?00?
-----------
  0000 0000
```

要清零数为1的位，本数对应位为0

a&0x0　a&0x1　a&0x8　a&0x9

a&0x40　a&0x41　a&0x48　a&0x49

4.2 位运算

4.2.3 位运算的应用

- 与(&)的应用
  - 取指定位

例：char a2=0xb6；现要求只保留低4位，而高4位清0，则：

1011 0110

& 0000 1111　　要保留的位，本数对应位为1

0000 0110

```
char a2=0xb6;
cout << "char a=0x" << hex << (int)a2
     << "  a&0x0F=" << dec << (a2&0x0F) << endl;
```

4.2 位运算

4.2.3 位运算的应用

- 或(|)的应用
  - 设定某些位为1

例：char a=0xb6；要求1、4位设为1，其它不变

1011 0110

| 0000 1001

1011 1111 (0xBF)

```
char a=0xb6;
cout << "a=" << hex << (int)a
     << " a|0x9=0x" << (a|0x9) << endl;
```

4.2 位运算

4.2.3 位运算的应用

- 异或(^)的应用
  - 特定位翻转（0，1互换）

例：char a1=0xb6; 高4位翻转，低4位不变

1011 0110

^ 1111 0000　　要翻转的位，本数对应位为1

0100 0110

```
char a1=0xb6;
cout << "a=" << hex << (int)a1
     << " a^0xF0=0x" << (a1^(char)0xF0) << endl;
```

4.2 位运算

4.2.3 位运算的应用

- 异或(^)的应用
  - 两数交换

```
char a=0xb6, b=0xc2;
cout << "a=" <<hex<<(int)a<<" b=" <<(int)b<< endl;
a = a^b; b = b^a; a = a^b;
cout << "a=" <<hex<<(int)a<<" b=" <<(int)b<< endl;
```

例：char a=0xb6,b=0xc2; 要求a，b互换

(1) a=1011 0110

b=1100 0010

a=0111 0100　　a=a^b=0x74

(2) b=1100 0010

a=0111 0100

b=1011 0110　　b=b^a=0xb6

(3) a=0111 0100

b=1011 0110

a=1100 0010　　a=a^b=0xc2

4.2 位运算

4.2.3 位运算的应用

综合应用例：班级有30个学生，老师每周都会对学生进行一次小测验，结果只有通过和不通过两种。为了更好地追踪测验的结果，我们用一个二进制位代表某个学生在一次测验中是否通过，显然全班的测验结果可以用一个无符号整数来表示：

```
unsigned long quiz1 = 0;    //位的集合
```

对序号为27的学生对应的位进行设置，以表示其通过了测验：

```
1UL << 27;    //生成一个值，该值只有第27位为1
quiz1 |= 1UL << 27;  //表示学生27通过了测验
```

4.2 位运算

4.2.3 位运算的应用

重新核对发现学生27实际上没有通过测试：

```
quiz1 &= ~(1UL << 27);  //学生27没有通过测验
```

检查学生27的测验情况：

```
bool status = quiz1 & (1UL << 27);  //学生27是否通过了测验?
```