# 计算机程序设计基础(1)
# --- C语言程序设计(13)

孙甲松
sunjiasong@tsinghua.edu.cn
电子工程系 信息认知与智能系统研究所
罗姆楼6-104
电话: 13901216180/62796193
2022.12.

# 期末机考安排

本学期期末机考全部采取线上考试的形式，共分为5个时间段：

周二（12月20日）中午：11:30开始准备，12:00开始，14:30结束。
周二（12月20日）下午：15:00开始准备，15:30开始，18:00结束。
周二（12月20日）晚上：18:30开始准备，19:00开始，21:30结束。
周三（12月21日）晚上：18:30开始准备，19:00开始，21:30结束。
周五（12月23日）晚上：18:30开始准备，19:00开始，21:30结束。

目前按照二级选课的时间段进行分组，具体分配情况请参考课程文件。如有课程冲突或其他有必要调换时间的特殊情况，请向**ybch14@163.com** 发送邮件，注明姓名、学号、原时间段、调整后时间段、需要调整的必要性理由。【时间调换截止时间本周三（12月14日）23:59（以收到邮件时间为准）】。请各位同学相互转告！之后将对每个场次建立微信群发布后续通知。

# 期末机考安排

每位同学只允许进行一次上机考试。如有同学进行多次考试，核实后，将按照所有考试成绩的最低分进行登记。

考试范围：截止到文件之前（不包括文件），其中数组、指针、字符串为重点，函数递归也可能会涉及。

# 期末机考安排

评分规则：

在2小时正式考试时间内完成并一次性通过验收者得满分20分。

在2小时到2小时15分钟内完成者，扣2分。

在2小时15分钟到2小时30分钟内完成者，扣4分。

2小时30分钟未完成者酌情评分。

此外，不论正式考试时间或延长时间，每次验收失败额外扣2分

考试纪律：

考试时，每个人自己独立完成考题的编程和调试。禁止对外求助，线上机考具体事宜会后续在微信群组说明。可以使用教材等参考书籍。请各位同学严格遵守清华大学教育教学纪律，一旦发现作弊等不端行为，将按照清华大学校规严肃处理。

# 第13章 位运算

## 13.1 二进制位运算

**位运算** 指对二进制位进行的运算。每个二进制位中只能存放0或1。通常，将一个数据用二进制数表示后，最右边的二进制位称为最低位（第0位），最左边的二进制位为最高位。

**C语言提供了六种位运算符**

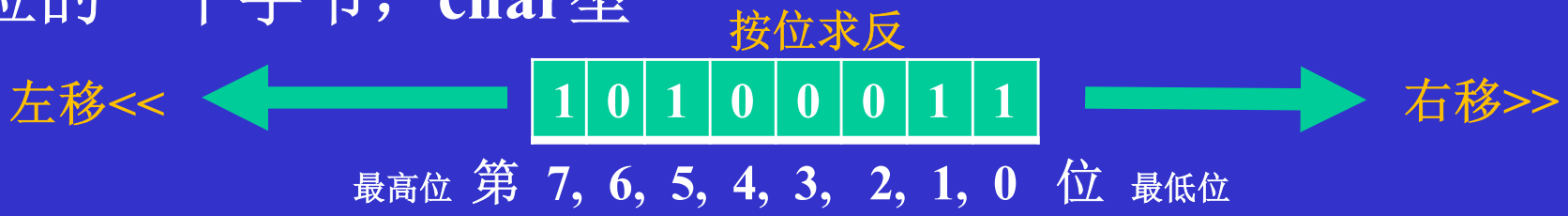
| 位运算符 | 意　义 |
|---|---|
| ~ | 按位取反 |
| & | 按位与 |
| \| | 按位或 |
| ^ | 按位异或 |
| << | 左移 |
| >> | 右移 |

**注意**

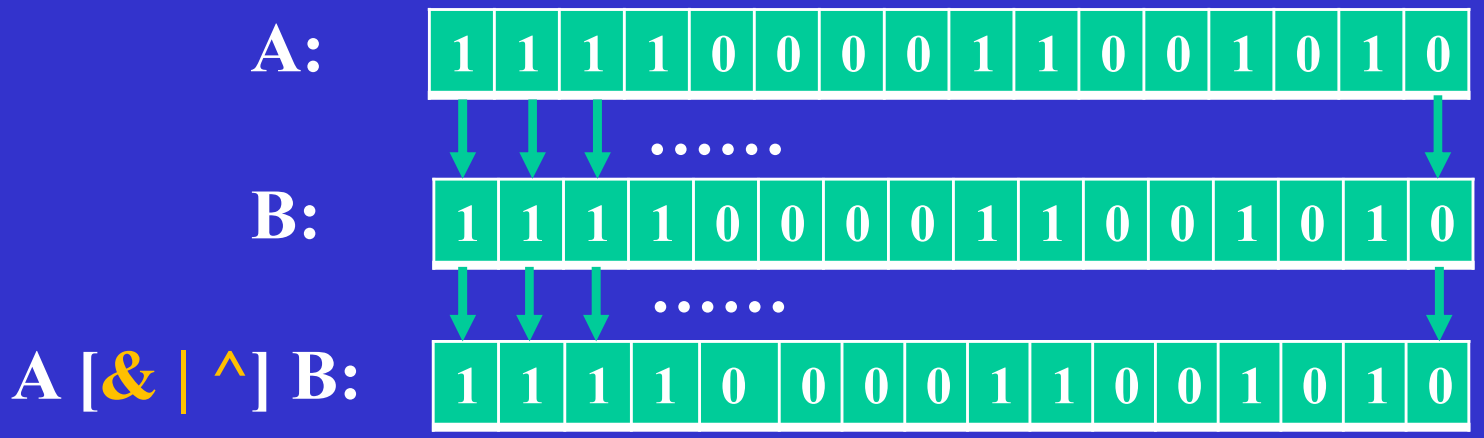
- 在这六种位运算符中，其中的按位取反~是*单目运算符*，只有一个运算对象，其他五个位运算符均为*双目运算符*，有两个运算对象。
- 位运算的运算对象只能是整型

(包括short, int, long, long long, unsigned的各种整数类型)或char及unsigned char型数据，但不能是实型数据。

## 8位的一个字节，char型

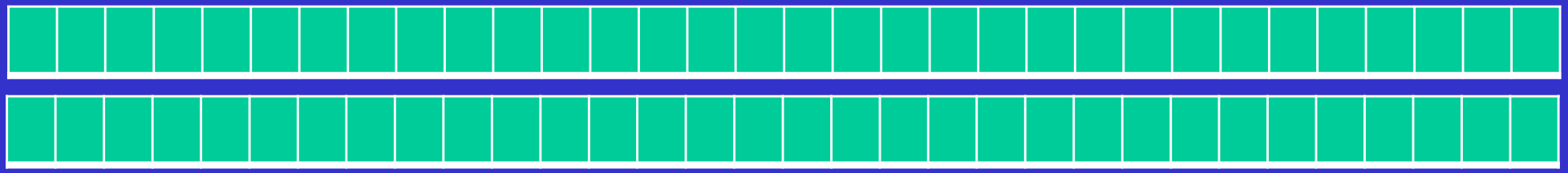



## 16位的一个short型，2个字节




## 32位的一个int或long型，4个字节

## 64位的一个超长long long型，8个字节



## 运算优先级

由高到低：逻辑非!　→

按位取反~　→

算术运算符（先*、/、%，后+、-）　→

左移运算符<<、右移运算符>>　→

关系运算符（6个）　→

按位与&、按位或|、按位异或^　→

逻辑与 &&　→

逻辑或 ||　→

赋值运算符 = 及其若干组合赋值运算符

总结：

① 按位取反~ 的运算优先级高于所有二目运算符。

② &、 | 和^ 的运算优先级高于&&，但低于关系运算符。

③ <<和>>的运算优先级高于关系运算符，但低于算术运算符。

④ 五个双目位运算符都可以与=组合成相应赋值位运算符：

&=、 |=、 ^=、 <<=、 >>=

## 13.1.1 “按位与”运算符（&）

### “按位与”运算符（&）

规则

- 若两个运算对象的对应二进制位均是1，则结果的对应位是1，否则为0。即对应二进制位上可能的“按位与”运算组合为：

**0&0=0，0&1=0，1&0=0，1&1=1**

例如，字符型数（假设一个字符型数占8位二进制位，下同）13（十六进制表示为0x0d）与字符型数21（十六进制表示为0x15）进行“按位与”如下：**13 & 21=5**，即

**0x0d & 0x15 = 0x05**

```
  0 0 0 0 1 1 0 1    13的二进制数
& 0 0 0 1 0 1 0 1    21的二进制数
-----------------
  0 0 0 0 0 1 0 1    运算结果 0x05
```

## 13.1.1 “按位与”运算符（&）

- 特别要指出的是，如果参加“按位与”运算的对象为负整数，则在计算机中是以补码形式表示的。

例如，负字符型数-13（二进制补码的十六进制表示为0xf3）与字符型数21（十六进制表示为0x15）进行“按位与”如下： **-13 & 21=17**，即 **0xf3 & 0x15 = 0x11**

| | | |
|---|---|---|
| | **1 1 1 1 0 0 1 1** | **-13**的二进制数补码表示 |
| **&** | **0 0 0 1 0 1 0 1** | **21**的二进制数补码（正数的补码是其本身） |
| | **0 0 0 1 0 0 0 1** | 运算结果 **0x11** |

## 功能

### （1）取出或判断数据中指定的位

例如，如果要取**short**型变量**x**的低字节（即低八位），则可以作如下运算： **x & 0x00ff**

如果要取短整型**short**变量**x**的高字节（即高八位），则可以作如下运算： **x & 0xff00**

例如， 判断**char**型变量**x**的第**3**位是否为**1**（最低位为第**0**位）

**if ( (x & 0x08) != 0)** 或者 **if (x & 0x08 )**

若条件表达式 **(x & 0x08) != 0** 的值为**1**（真），则表示**x**的第**3**位为**1**，否则表示**x**的第**3**位为**0**。

特别要注意：由于“按位与”运算符“**&**”的优先级要低于关系运算符“**!=**”，因此，**if ( (x & 0x08) != 0)** 表达式中 **(x & 0x08)** 外面的一对圆括号不能省略。

- 判断某一位是否为1，经常用来判断是否满足某个特性。

比如，判断一个从文件目录中读入的某文件属性是否是子目录：

```
if (Win32_Find_Data.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
```

此语句绝对不能写成：

```
if (Win32_Find_Data.dwFileAttributes == FILE_ATTRIBUTE_DIRECTORY)
```

因为一个文件的属性的包含多种情况，比如，可能

Win32_Find_Data.dwFileAttributes = 0x00000011

表示此文件既是目录

（FILE_ATTRIBUTE_DIRECTORY=0x00000010）

又是只读的

（FILE_ATTRIBUTE_READONLY=0x00000001）

## （2）将数据中的指定位清零

例如，要将短整型short变量x的低字节（即低八位）清零，则可以用如下语句：

**x=x & 0xff00;　或　x &= 0xff00;**

这里0xff00 被称为掩码（mask）。如果要将短整型short变量x的高字节（即高八位）清零，则可以用如下语句：

**x=x & 0x00ff;　或　x &= 0x00ff;**

如果要将整型int变量x的高16位清零，则可以用如下语句：

**x=x & 0x0000ffff;　或　x &= 0x0000ffff;**

又如，为了将短整型short变量x的第4位清零，则可以用如下语句：

**x=x & 0xffef;　　或　　x &= 0xffef;**

- 注意：在某些情况下，&可替代逻辑&&：

  比如：　**if ((a==0) && (b==0))**

  写作：　**if ((a==0) & (b==0))**

是等价的。

但前一个执行效率更高，因为**a**不为**0**时，不再判断**b==0**。

【例13-1】 编制一个C程序，其功能是将正整型数组中所有元素转换为不大于它的最大偶数，并显示输出。

为了将一个正整数转换为不大于它的最大偶数，只需将该正整数所对应的二进制数的最低位清零即可，即用**0xfffffffe**与该正整数作“按位与”运算。其C程序如下：

```
#include  <stdio.h>
main( )
{ int  k，  a[10]={23,14,24,31,46,55,33,68,27,40};
   for (k=0; k<10; k++)
      printf("％5d"，  a[k]);
   printf("\n");
   for (k=0; k < 10; k++)
      a[k] &= 0xfffffffe; /*将正整数转换为不大于它的最大偶数*/
   for (k=0; k < 10; k++)
      printf("％5d"，  a[k]);
   printf("\n");
}
```

程序的运行结果为：

**23  14  24  31  46  55  33  68  27  40**

**22  14  24  30  46  54  32  68  26  40**

## 13.1.2 “按位或”运算符（|）

### “按位或”运算符（|）

规则

- 若两个运算对象的对应二进制位中有一个是1，则结果的对应位是1，否则为0。即对应二进制位上可能的“按位或”运算组合为：

**0|0=0，0|1=1，1|0=1，1|1=1**

例如， **char整数13**（十六进制表示为**0x0d**）与**char整数21**（十六进制表示为**0x15**）进行“按位或”如下：

```
  0 0 0 0 1 1 0 1   13的二进制数
| 0 0 0 1 0 1 0 1   21的二进制数
-----------------
  0 0 0 1 1 1 0 1   运算结果 0x1d
```

因此 **13 | 21 = 29**，即 **0x0d | 0x15 = 0x1d**

## 13.1.2 “按位或”运算符（|）

又如，**char**型数**-13**（二进制补码的十六进制表示为**0xf3**）与**char**型数**21**（十六进制表示为**0x15**）进行“按位或”。

| | |
|---|---|
| **1 1 1 1 0 0 1 1** | **-13**的二进制数补码表示**0xf3** |
| **\| 0 0 0 1 0 1 0 1** | **21**的二进制数补码（正数的补码是其本身） |
| **1 1 1 1 0 1 1 1** | 运算结果 **0xf7** |

因此，**-13 | 21 = -9**，即 **0xf3 | 0x15 = 0xf7**

- “按位或”运算通常用于把一个数据的某些位强置为1，而其余位不变。

例如，要将short 型变量x的低字节（即低八位）置1，而高字节（即高八位）不变，则可以用如下赋值语句：

**x = x | 0x00ff;** 或 **x |= 0x00ff;**

如果要将short型变量x的高字节置1，而低字节不变，则可以用如下赋值语句：

**x = x | 0xff00;** 或 **x |= 0xff00;**

又如，为了将int型变量x的第0位与第4位强置1，而其余位不变，则可以用如下赋值语句：

**x = x | 0x00000011;** 或 **x |= 0x00000011;**

- 注意：在某些情况下，| 可替代逻辑或 || ：

  比如：　**if ((a==0) || (b==0))**

  写作：　**if ((a==0) | (b==0))**

是等价的。

但前一个执行效率更高，因为**a**为**0**时，不再判断**b==0**。

【例13-2】 编制一个C程序，其功能是将正整型数组中所有元素转换为不小于它的最小奇数，并逐个输出。

为了将一个正整数转换为不小于它的最小奇数，只需将该正整数所对应的二进制数的最低位置1即可，即用**0x00000001**与该正整数作“按位或”运算。其C程序如下：

注意：**a[k] |= 0x00000001;** 不等价于 **a[k] += 0x00000001;**

因为后者可能把奇数变为偶数。

```
#include  <stdio.h>
main( )
{   int k, a[10]={23,14,24,31,46,55,33,68,27,40};
    for (k=0; k<10; k++)
        printf("％5d", a[k]);
    printf("\n");
    for (k=0; k<10; k++)
        a[k] |= 0x00000001; /*将正整数转换为不小于它的最小奇数*/
    for (k=0; k<10; k++)
        printf("％5d",a[k]);
    printf("\n");
}
```

程序的运行结果为：

**23  14  24  31  46  55  33  68  27  40**

**23  15  25  31  47  55  33  69  27  41**

## 13.1.3 “按位异或”运算符（^）

### “按位异或”运算符（^）

规则

- 若两个运算对象的对应二进制位不相等，则结果的对应位是1，否则为0。即对应二进制位上可能的“按位异或”运算组合为：

**0^0=0，0^1=1，1^0=1，1^1=0**

---

例如， **char型数13**（十六进制表示为**0x0d**）与**char型数21**（十六进制表示为**0x15**）进行“按位异或”如下：

```
  0 0 0 0 1 1 0 1    13的二进制数
^ 0 0 0 1 0 1 0 1    21的二进制数
-------------------
  0 0 0 1 1 0 0 0    运算结果 0x18
```

因此，**13 ^ 21 = 24**，即 **0x0d ^ 0x15 = 0x18**

按位异或^又被称为：“不进位加”

## 13.1.3 “按位异或”运算符（^）

又如，**char**型数**-13**（二进制补码的十六进制表示为**0xf3**）与**char**型数**21**（十六进制表示为**0x15**）进行“按位异或”如下：

| | | |
|---|---|---|
| | 1 1 1 1 0 0 1 1 | -13的二进制数补码表示0xf3 |
| ^ | 0 0 0 1 0 1 0 1 | 21的二进制数补码0x15(正数的补码是其本身) |
| | 1 1 1 0 0 1 1 0 | 运算结果 0xe6 |

因此，-13 ^ 21 = -26，即 0xf3 ^ 0x15 = 0xe6

## 性质

（1） 使数据中的某些位取反，即将0变为1，1变为0。

例如，要将short型变量x的低字节（即低八位）按位取反，而高字节（即高八位）不变，则可以用如下赋值语句： **x=x^0x00ff;** 或 **x ^= 0x00ff;**

又如，为了将short型变量x的第0位与第4位取反，而其余位不变，则可以用如下赋值语句：

**x=x^0x0011;** 或 **x ^= 0x0011;**

（2）同一个数据进行异或运算后，其结果为0。利用异或运算的这个性质，可以将变量清零。

例如，为了将int型变量x清零，则可以用如下赋值语句： **x=x^x;** 或 **x ^= x;**

## 性质

（3）设x与y均为整型数据，则有 (x^y)^y=x

利用“按位异或”的这个性质，可以实现不借助于第三者交换两个整型变量的值：

```
x=x^y;    y=x^y;    x=x^y;
```

```
#include <stdio.h>
#pragma warning(disable:4996)
int main( )
{  int x,y;
   scanf("%d%d",&x,&y);
   printf("x=%d y=%d\n",x,y);
   x=x^y;   y=x^y;   x=x^y;
   printf("x=%d y=%d\n",x,y);
}
```

程序的运行结果为：

**3 5**

**x=3 y=5**

**x=5 y=3**

请按任意键继续. . .

## 13.1.4 “按位取反”运算符（~）

### “按位取反”运算符（~）

规则

- 将运算对象中的各二进制位值取反，即将0变为1，1变为0。对应二进制位上可能的“按位取反”运算组合为：

~0=1，~1=0

例如， 对char型数13（十六进制表示为0x0d）进行“按位取反”运算如下：

```
~   0 0 0 0 1 1 0 1   13的二进制数
-----------------
    1 1 1 1 0 0 1 0
```

因此，~13= -14，即 ~0x0d=0xf2

注意：不要把“按位取反”运算~与逻辑非运算!混为一谈。

~按位取反运算符是把运算对象的每一位求反。1变为0，0变为1。

!运算符是把整个运算对象的值求非。值不为0变为0，值为0变为1。

## 13.1.5 “左移”运算符（<<）

### 左移”运算符（<<）

规则

- 将运算对象中的每个二进制位向左移动若干位，从左边移出去的高位部分被丢弃，右边空出的低位部分补0。

例如：若int x=3；x = x << 3； 则x值为24，

相当于 x=x*8；8=$2^3$

x = x << 3；也可以写为： x <<= 3；

一般来说，将整数左移k位，相当于将该整数乘以$2^k$。

## 13.1.6 “右移”运算符（>>）

### “右移”运算符（>>）

规则

- 将运算对象中的每个二进制位向右移动若干位，从右边移出去的低位部分被丢弃。但左边空出的高位部分是补0还是补1，要视下列具体情况而定：

（1）若右移对象为无符号整型数，则右移后左边空出的高位部分补0。

（2）若右移对象为一般整型数或字符型数据，当该数据的最高位为0（对于一般整型来说即为正数），则右移后左边空出的高位部分补0。当该数据的最高位为1（对于一般整型来说即为负数），则与使用的编译系统有关。有的编译系统将右移后左边空出的高位部分补1，称为“算术右移”；有的编译系统将右移后左边空出的高位部分补0，称为“逻辑右移”。32位编译系统VS2008及其后续的VS系列编译器，都属于“算术右移”。

例如：若**int x=13；x = x >> 2；**则**x**值为**3**，

相当于 **x=x/4；4=2$^2$**

**x = x >> 2；** 也可以写为： **x >>= 2；**

一般来说，将整数算术右移**k**位，相当于将该整数除以**2$^k$** 。

## 13.2 位段（bit-field）

- 又被称为位域，定义位段结构类型的一般形式为：

```
struct 位段结构类型名
{ 成员表 };
```

例如：

```
struct packed_d
{ unsigned short f1:2;
  unsigned short f2:1;
  unsigned short f3:3;
  unsigned short f4:2;
  unsigned short f5:5;
};
```

- 定义了一个位段结构类型，名为**packed_d**，共包含**5**个成员（又称为位段），每个成员均为无符号**short**类型，其中成员**f1**占**2**个二进制位，**f2**占**1**个二进制位，**f3**占**3**个二进制位，**f4**占**2**个二进制位，**f5**占**5**个二进制位。至于这些位段在存储单元中的具体存放位置，是由编译系统来分配的，一般用户不必考虑。各位段在存储单元中一般是从右到左（即从低位到高位）顺序分配的。

- 上述定义的位段结构类型需要占**2**个字节（即**16**个二进制位），其存储结构为：

| 15 13 | 12 8 | 7 6 | 5 3 | 2 1 | 0 |
|---|---|---|---|---|---|
| 空 | f5 | f4 | f3 | f2 | f1 |

- 定义了位段结构类型后，就可以定义位段结构类型的变量。

  例如，　**struct packed_d x，y;**

  定义了属于位段结构类型packed_d的两个变量x与y。

- 与定义结构体类型变量一样，不仅可以将位段结构类型与该类型的变量分开定义，也可以在定义位段结构类型的同时定义该类型的变量，例如，

```
struct  packed_d
{ unsigned short f1:2;
  unsigned short f2:1;
  unsigned short f3:3;
  unsigned short f4:2;
  unsigned short f5:5;
} x, y;
```

- 还可以直接定义无名的位段结构类型的变量，例如：

```
struct
{ unsigned short f1:2;
  unsigned short f2:1;
  unsigned short f3:3;
  unsigned short f4:2;
  unsigned short f5:5;
} x, y;
```

- 对位段结构成员的引用方式，与引用一般结构体成员的方式相同。例如，

```
x.f4=3;
```

表示将3赋给位段结构类型变量x的位段（即成员）f4中。

- 但必须注意，在对位段进行赋值时，要考虑到该位段所占用的二进制位数，如果所赋的数值超过了位段的表示范围，则自动取其低位数字。例如，

```
x.f4=5;
```

由于f4位段只占2个二进制位，因此，实际赋给f4位段的是5的二进制表示（即101）中的低2位，也就是1。

- 在定义位段与使用位段时，要注意以下几个问题：

（1）位段成员的类型必须是unsigned型。

（2）在位段结构类型中，可以定义无名位段，这种无名位段具有位段之间的分隔（或占位）作用。例如，

```
struct  packed_data
{ unsigned char f1:2;
  unsigned char f2:1;
  unsigned char :2;
  unsigned char f3:3;
};
```

在这个位段结构定义中的第3个位段（成员）是无名位段，它占有2个二进制位，在位段f2与f3之间起分隔（或占位）作用。无名位段所占用的空间不起作用。

如果无名位段的宽度值为0，则表示下一个位段从一个新的字节开始存放。例如，

```
struct  packed_data
{ unsigned char f1:2;
  unsigned char f2:1;
  unsigned char :0;
  unsigned char f3:3;
};
```

这个位段结构要占2个字节。

(3) 每个位段（成员）所占的二进制位数一般不能超过编译器的一个字长（比如32位或64位）。

(4) 位段不能说明为数组，也不能用指针指向位段成员。

例如，

```
struct {
  unsigned char  a:3;
  unsigned char  b:2;
  unsigned char  c:3;
} byte;
byte.a=07; byte.b=02; byte.c=06;
char *p;   p=&byte.a;
```

编译时，会出现错误信息：

error C2104: 位域上的“&”被忽略

**(5)** 不能用 sizeof 求段位成员的大小。

例如，

```
struct {
    unsigned char  a:3;
    unsigned char  b:2;
    unsigned char  c:3;
} byte;
byte.a=07; byte.b=02; byte.c=06;
sizeof(byte.a);
```

编译时，会出现错误信息：

error C2070: “char”: 非法的sizeof 操作数

**(6)** 在位段结构类型定义中，可以包含非位段成员。例如，

```
struct  packed_x
{ int  n;
  unsigned  int f1:2;
  unsigned  int f2:1;
  unsigned  int f3:2;
};
```

其中**n**为非位段成员，它单独占**4**个字节。整个**packed_x**结构体将占**2**个**int**共**8**个字节。

非位段成员也可以放在两个位段成员之间，例如，

```
struct  packed_x
{ unsigned  int  f1:2;
  int  n;
  unsigned  int f2:1;
  unsigned  int f3:2;
};
```

非位段成员**n**在位段成员**f1**与**f2**之间。由于**n**单独占**4**个字节，因此整个**packed_x**结构体将占**3**个**int**共**12**个字节。

非位段成员也可以在所有位段成员之后，例如，

```
struct  packed_x
   { unsigned  int f1:2;
     unsigned  int f2:1;
     unsigned  int f3:2;
     int  n;
   }
```

非位段成员**n**在位段成员**f1**、**f2**与**f3**之后。整个**packed_x**结构体将占**2**个**int**共**8**个字节。

但不管位段成员在两个位段成员之间，或非位段成员在所有位段成员之后，非位段成员总是从下一个字节开始存放，当前字节剩下的位空间不再使用。

非位段成员的引用方式与普通结构体成员的引用方式完全相同。

**(7)** 位段结构体类型变量中的位段成员可以在一般的表达式中被引用，并被自动转换为相应的整数。

例如，下列赋值语句是合法的：　　p=x.f4 + 2*x.f2;

## 13.3 程序举例

【例13-3】 编写一个C程序，其功能是：从键盘输入一个无符号整数m以及位移位数n，当n＞0时，将m循环右移n位，当n<0时，将m循环左移|n|位。

将一个无符号整数m循环移n位的方法如下：

- 首先用sizeof( )函数确定一个无符号整数所占的二进制位数k。
- 如果是循环右移，则先将m右移n位（即将原数的高k-n位移到低位），再将m左移k-n位（即将原数的低n位移到高位），然后将它们做按位或运算（即将它们合并）。
- 如果是循环左移，则先将m左移n位（即将原数的低k-n位移到高位），再将m右移k-n位（即将原数的高n位移到低位），然后将它们做按位或运算（即将它们合并）。

```
unsigned int moveright(unsigned int m,int n)
{  unsigned int z;
   int k;
   k=8*sizeof(unsigned int);
   z=(m>>n) | (m<<(k-n));
   return z;
}
unsigned int moveleft(unsigned int m,int n)
{  unsigned int z;
   int k;
   k=8*sizeof(unsigned int);
   z=(m<<n) | (m>>(k-n));
   return z;
 }
```

将m循环右移n位

将m循环左移n位

```
#include  <stdio.h>
main( )
{ unsigned int m;
   int n;
   printf("input m:");
   scanf("%x", &m);
         /*以十六进制格式输入要位移的无符号整数 */
   printf("input n:");
   scanf("%d", &n);  /*输入位移量*/
   if (n>0)  /* 循环右移 */
         printf("moveright=%x\n", moveright(m, n));
   else   /* 循环左移 */
         printf("moveleft=%x\n", moveleft(m, -n));
}
```

程序运行结果（十六进制格式）：

input m:ABCDEFAB

input n:8

moveright=ABABCDEF

input m:ABCDEFAB

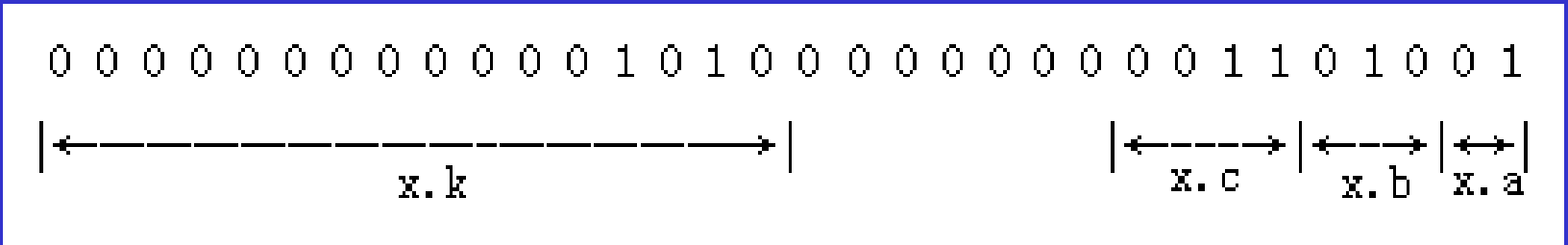
input n:-8

moveleft=CDEFABAB

【例13-4】 设位段的空间分配是从右到左的（即从低位到高位），指出下列C程序的输出结果：

```
#include <stdio.h>
struct packed_bit
{ unsigned short a:2;
   unsigned short b:3;
   unsigned short c:4;
   short  k;
} x;
main( )
{ x.a=1;  x.b=2;  x.c=3;  x.k=10;
   printf("%x\n", x);
   printf(“sizeof(x)= %d\n", sizeof(x) );
   printf("%x\n", x.k);
   printf("sizeof(x.k)= %d\n", sizeof(x.k) );
}
```

在上述程序的位段结构类型中，定义了三个位段**a**，**b**，**c**，以及非位段成员**k**，当为位段结构类型变量**x**中的各成员赋值后，它们在计算机内存中被分配的存储单元以及为各位段赋值后的状态如下图所示（右边为低位，左边为高位）：




- 第一个输出语句

```
printf("%x\n", x);
```

中的格式说明符为整型格式说明符**%x**，而输出项为位段结构体类型变量(**x**是一个无符号的**int**型变量)，其十进制值为**105**，即十六进制值输出为：**a0069**

- 第二个执行语句

```
printf("sizeof(x)= %d\n", sizeof(x) );
```

其输出项为位段结构体类型变量的大小，**x**中位段**a,b,c**共**9**位，可以存放在一个**unsigned short**中，同非位段**short**成员**k**，组成一个无符号的**int**型变量。因此输出结果为:

```
sizeof(x)= 4
```

- 第三个执行语句

```
printf("%x\n", x.k);
```

是按整型格式说明符**%x**输出构体类型变量**x**中**short**型成员**x.k**的值，即输出为：**a**

- 第四个执行语句

```
printf("sizeof(x.k)= %d\n", sizeof(x.k) );
```

其输出项为位段结构体类型变量的同非位段**short**成员**k**。因此输出结果为：　sizeof(x.k)= 2

- 因此，该程序运行的全部输出结果为：

a0069

sizeof(x)= 4

a

sizeof(x.k)= 2

C语言到此结束！

C++在向你招手！