
程序设计专题

主讲教师: 纪守领

学院: 计算机科学与技术

NESA Lab: <https://nesa.zju.edu.cn/>

Email: sji@zju.edu.cn



专题五：位运算和文件进阶

- 位运算

- 文件的操作和应用

位运算

- 位运算是指进行二进制位的运算。在系统软件中，常要处理二进制位的问题。C语言提供了按位运算的功能，这使得它与其他高级语言相比，具有很强的优越性。
- 相比算数运算，位运算在计算机中运算效率更高，巧妙地采用位运算能够产生相当高效的程序。
- C语言提供了六种位运算符：

类 型	运算符	含义
位逻辑	&	按位与
		按位或
	^	按位异或
运算符	~	取反
移位运算符	<<	左移
	>>	右移

位运算使用的注意事项

- 位运算的操作数只能是整型或字符型的数据以及它们的变体，不能为实型或结构体等类型的数据。
- 操作数的位运算不改变原操作数的值。
- 六个位运算符的优先级由高到低依次为：取反、左移和右移、按位与、按位异或、按位或。
- 两个不同长度的数据进行位运算时，系统会将二者按右端对齐。

按位与运算 &

- 参与运算的两数（以补码方式）各对应的二进制位相与，只有对应的两个二进制位均为1时，结果位才为1，否则为0，是双目运算符。
- 例如（8位二进制数据类型）：

9 & 5

9的二进制补码： 0 0 0 0 1 0 0 1

5的二进制补码： 0 0 0 0 0 1 0 1

运算结果： 0 0 0 0 0 0 0 1（1的二进制补码）

按位与运算 & 的应用

□ 清零

- 按位与运算通常用来对某些位清0。由按位与的规则可知：为了使某数的指定位清零，可将该数按位与某一特定数。该数中为1的位，特定数相应位应为0；该数中为0的位，特定数中相应位可以为0也可以为1。由此可见，能对某一个数的指定位清零的数并不唯一。

□ 取一个数中某些位

- 可将该数与一个特定数进行&运算，对于要取的那些位，特定数中相应位设为1。

□ 取出数中某一位

- 要想将一个数的某一位保留下来，可将该数与一个特定数进行&运算，特定数的相对应的那位应为1。

按位与运算 & 的应用-清零

□ 计算00110110对16取余的结果。

原数的二进制补码: 0 0 1 1 0 1 1 0

15的二进制补码: 0 0 0 0 1 1 1 1

运算结果: 0 0 0 0 0 1 1 0 (6的二进制补码)

程序可写为:

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    unsigned char a = 0x36, b = 0x0f, c;
```

```
    c = a & b;
```

```
    printf("a = %hhx, b = %hhx, c = %hhx", a, b, c);
```

```
}
```

运行结果:

a = 36, b = f, c = 6

按位与运算 & 的应用-取某些位

- 计算输入整数a除以16的结果。即清除低4位。

可作 $a \& 240$ (240的二进制为11110000)

程序可写为：

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    unsigned char a, b = 0xf0, c;
```

```
    scanf("%hhu", &a);
```

```
    c = a & b;
```

```
    printf("a = %hhx, b = %hhx, c = %hhx", a, b, c);
```

```
}
```

运行结果：

117<Enter>

a = 75, b = f0, c = 70

按位与运算 & 的应用-取某一位

- 判断输入的整数的奇偶性。

取输入整数的最低位，最低位为1则为奇数，最低位为0则为偶数。

程序可写为：

```
#include <stdio.h>
int main()
{
    unsigned char a;
    scanf("%hhu", &a);
    if(a & 0x01)
        printf("a is odd.");
    else
        printf("a is even.");
}
```

运行结果：

```
1<Enter>
a is odd.
```

按位或运算 |

- 参与运算的两数（以补码方式）各对应的二进制位相或，只有对应的两个二进制位有一个为1时，结果位就为1。它是双目运算符。
- 例如：

9 | 5

9的二进制补码： 0 0 0 0 1 0 0 1

5的二进制补码： 0 0 0 0 0 1 0 1

运算结果： 0 0 0 0 1 1 0 1 (13的二进制补码)

按位或运算 | 的应用-判断负数

- 判断连续的n个整数是否存在一个负数。

只需判断n个整数的按位或是否为负数

程序可写为：

```
#include <stdio.h>
int main()
{
    int a = -1, b = 1, c = 2, d = 3;
    printf("%d", (a | b | c | d) < 0);
}
```

运行结果：

1

不高效的写法：

```
printf("%d", a < 0 | b < 0 | c < 0 | d < 0);
```

按位异或运算 ^

- 参与运算的两数（以补码方式）各对应的二进制位相异或，当对应的两个二进制位不同时，结果位就为1，否则为0。它是双目运算符。
- 例如：

$9 \wedge 5$

9的二进制补码： 0 0 0 0 1 0 0 1

5的二进制补码： 0 0 0 0 0 1 0 1

运算结果： 0 0 0 0 1 1 0 0 (12的二进制补码)

- 一个数与自身的异或为0。0与任何数异或都为原数。

按位异或运算 ^ 的应用

- 使特定位翻转
 - 要使哪几位翻转就将其与进行按位异或运算的数的相应位置设置为1。
- 使特定位保留原值
 - 要使哪几位保留原值就将其与进行按位异或运算的数的相应位置设置为0。
- 交换两个数，不用临时变量
- 寻找单身狗

按位异或运算 ^ 的应用-互换数值

□ 设有整数 $a=5$, $b=7$ 。编写程序利用异或运算将 a , b 的值互换。

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

因为 \wedge 运算符的结合方向是从右到左, 因此上述语句等价于三个连续语句:

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

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

按位异或运算 ^ 的应用-互换数值

□ 记初始的a, b为 a_0, b_0 , 证明:

第一步: 执行 $a = a \oplus b$

$$a_1 = a_0 \oplus b_0$$

第二步: 执行 $b = b \oplus a$

$$\begin{aligned} b_1 &= b_0 \oplus a_1 = b_0 \oplus (a_0 \oplus b_0) \text{ (异或运算有交换律和结合律)} \\ &= b_0 \oplus b_0 \oplus a_0 = a_0 \text{ (变量b获得了变量a原本的值)} \end{aligned}$$

第三步: 执行 $a = a \oplus b$

$$\begin{aligned} a_2 &= a_1 \oplus b_1 = a_0 \oplus b_0 \oplus a_0 \text{ (异或运算有交换律)} \\ &= a_0 \oplus a_0 \oplus b_0 = b_0 \text{ (变量a获得了变量b原本的值)} \end{aligned}$$

按位异或运算 ^ 的应用-寻找单身狗

- 一群人出游，其中只有一条单身狗。每个人有自己的编号，且情侣的编号相同，请问单身狗的编号是多少？

即：一堆整数，其中只有一个整数只出现了1次，其余整数均出现2次，找出只出现了1次的整数。

考虑到任何数与自身异或均为0，且0与任何数异或均为原数。因此所有人的编号异或之后的结果即为单身狗的编号。。。

按位取反运算 ~

- 参与运算的数（以补码方式）各对应的二进制位取反，即0变1，1变0。它是单目运算符。
- 例如：

~9

9的二进制补码： 0 0 0 0 1 0 0 1

运算结果： 1 1 1 1 0 1 1 0 (-10的二进制补码)

- 适当的使用可以增加程序的移植性。如要将整数a的最低位置为0，我们通常采用 $a = a \& \sim 1$ 来完成，因为这样对a是16位数还是32位数均不受影响。

左移运算 <<

- 把<<左边的运算数的各二进制位全部左移若干位，由<<右边的数指定移动的位数，**高位丢弃，低位补0**。
- 例如：

a为十进制3， $a \ll 4$

a的二进制补码： 0 0 0 0 0 0 1 1

运算结果： 0 0 1 1 0 0 0 0 (48的二进制补码)

- 左移1位相当于该数乘以2；左移n位相当于该数乘以 2^n 。但此结论只适用于该数左移时被溢出舍弃的高位中不包含1的情况。
- 左移比乘法运算快得多，有的C编译器自动将乘2运算用左移一位来实现。

右移运算 >>

- 把>>左边的运算数的各二进制位全部右移若干位，由>>右边的数指定移动的位数。
- 右移1位相当于该数除以2；右移n位相当于该数除以 2^n 。
- 对于有符号数，在右移时，符号位将随同移动。当为整数时，最高位补0；而为负数时，符号位为1，最高位是补0还是补1取决于计算机系统的规定。移入0的称为“逻辑右移”；移入1的称为“算术右移”。我们可以通过编写程序来验证所使用的计算机系统是采用的哪种方式。很多系统规定为补1，即“算术右移”。
- 例如：
 a为十进制-2，a >> 1
 a的二进制补码： 1 1 1 1 1 1 1 0
 算术右移： 1 1 1 1 1 1 1 1 (-1的二进制补码)
 逻辑右移： 0 1 1 1 1 1 1 1 (127的二进制补码)

学生成绩文件统计

- 例12-1 有5位学生的计算机等级考试成绩被事先保存在数据文件 f12-1.txt（需事先准备好该文件，放在代码同目录下）中，包括学号、姓名和分数，文件内容如下：

301101 Zhangwen 91

301102 Chenhui 85

301103 Wangweidong 76

301104 Zhengwei 69

301105 Guowentao 55

- 请读出文件的所有内容显示到屏幕，并输出平均分

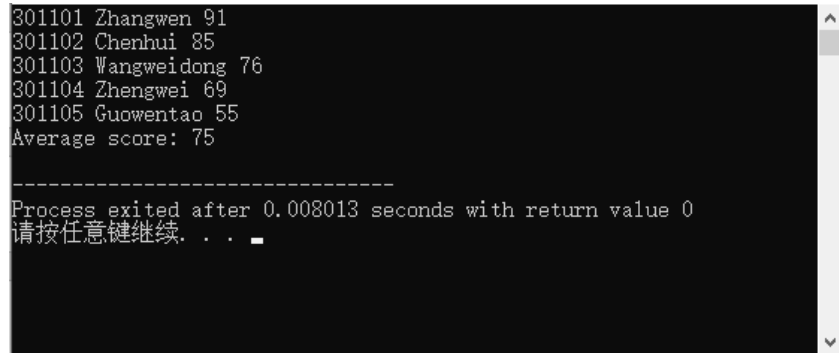
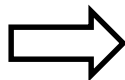
学生成绩文件统计



A screenshot of a Windows Notepad window titled "f12-1.txt - 记事本". The window contains the following text:

```
301101 Zhangwen 91  
301102 Chenhui 85  
301103 Wangweidong 76  
301104 Zhengwei 69  
301105 Guowentao 55
```

The status bar at the bottom indicates "第 1 行, 第 1" (Line 1, Column 1), "100%", "Windows (CRLF)", and "UTF-8".



A screenshot of a terminal window showing the output of a program. The output is as follows:

```
301101 Zhangwen 91  
301102 Chenhui 85  
301103 Wangweidong 76  
301104 Zhengwei 69  
301105 Guowentao 55  
Average score: 75  
-----  
Process exited after 0.008013 seconds with return value 0  
请按任意键继续. . .
```

源程序

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    FILE *fp;                                /*1.定义文件指针*/
    long num;
    char stname[20];
    int score;
    int i, avg_score = 0;

    if((fp = fopen("f12-1.txt","r")) == NULL)    /*2.打开文件*/
    {
        printf("File open error!\n");
        exit(0);
    }
```

源程序

```
/*3.文件处理（逐个读入和处理数据）*/
for(i = 0; i < 5; i++)
{
    /*从文件读入成绩保存到变量*/
    fscanf(fp, "%ld%s%d", &num, stname, &score);

    avg_score += score;    /*统计总分*/
    /*输出成绩到屏幕*/
    printf("%ld %s %d\n", num, stname, score);
}

/*输出平均分到屏幕*/
printf("Average score: %d\n", avg_score / 5);
if(fclose(fp)){
    printf("Can not close the file!\n");
    exit(0);
}
return 0;
}

/*4.关闭文件*/
```

文件的概念

- **文件**：操作系统中的文件是指驻留在外部介质（如磁盘等）中的一个有序数据集。
- **各种类型的文件**
 - **程序文件**：源文件、目标程序、可执行程序
 - **数据文件（输入/输出）**：文本文件、图像文件、声音文件、可执行文件等
- **文件的特点**
 - 数据**永久保存**；数据**长度不定**；数据按**顺序存取**

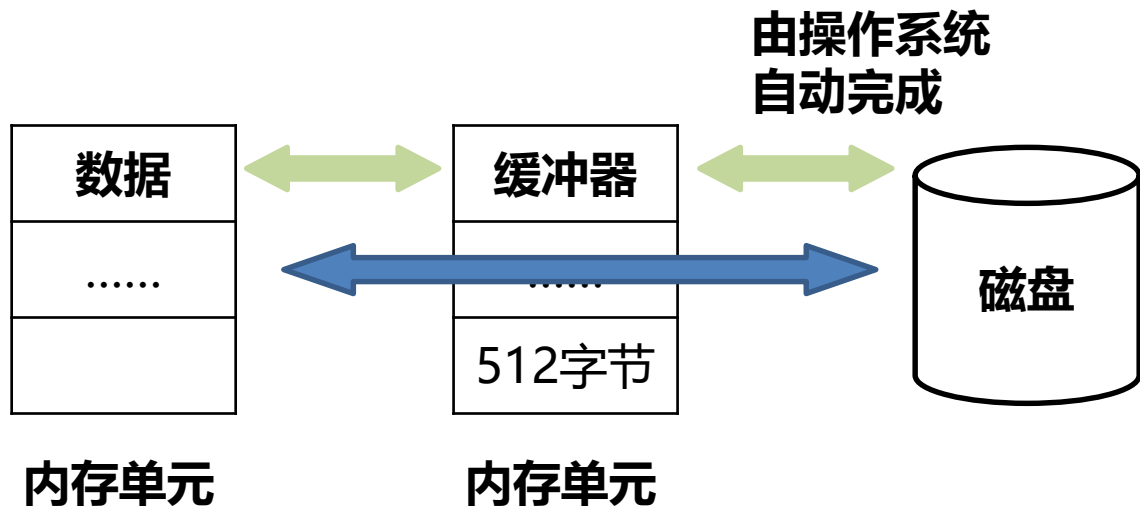
文本文件和二进制文件

字节	字节	字节	字节	字节	字节
----	----	----	----	----	----	-------

- C语言中的文件是数据流（由一个个的字节数据组成）
- 文件的两种数据形式：
 - ASCII码（文本文件text stream） 字符流
 - 二进制码（二进制文件binary stream） 二进制流
- 二进制文件是直接把内存数据以二进制形式保存，例如整数1234
 - 文本文件保存：49 50 51 52（4个字符）
 - 二进制文件保存：04D2（1234的二进制数）

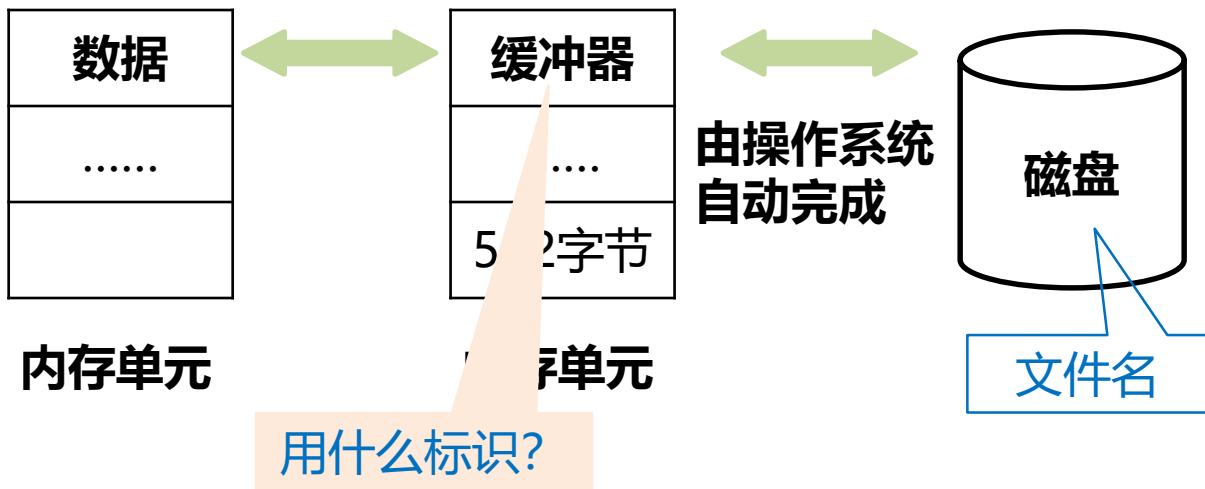
缓冲文件系统

□ 磁盘访问速度与内存访问速度差别很大，直接访问磁盘效率很低



缓冲文件系统

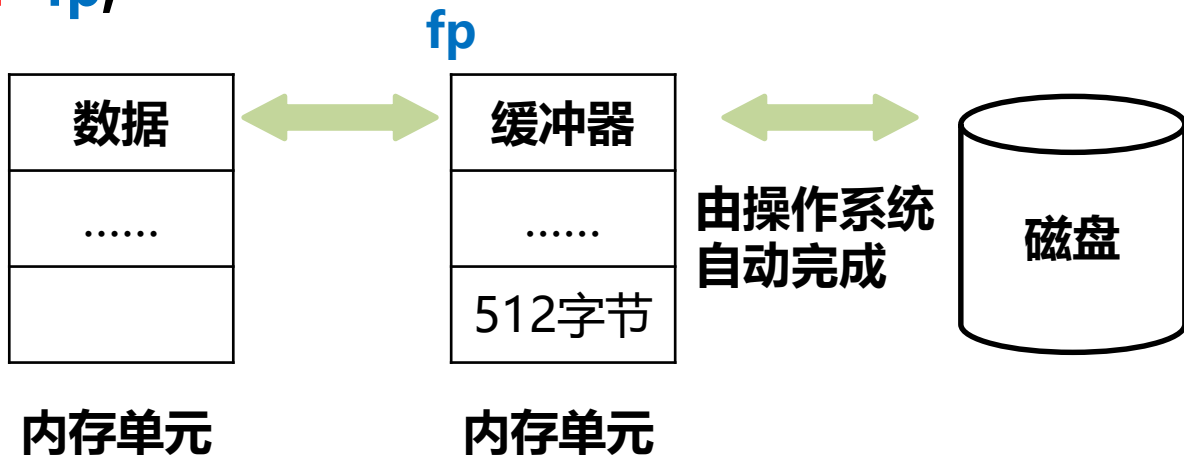
- 向磁盘输出数据：数据 ----> 缓冲区，装满缓冲区后 ----> 磁盘文件。
- 从磁盘读取数据：先**一次性**从磁盘文件将**一批数据输入**到缓冲区，然后再从缓冲区**逐个**读入数据到变量。



缓冲文件系统与文件类型指针

□ 用文件指针指示文件缓冲区中具体读写的位置

FILE *fp;



□ 同时使用多个文件时，每个文件都有缓冲区，用不同的文件指针分别指示。

文件结构与文件类型指针

□ 文件结构与自定义类型typedef

FILE: 结构类型, 用typedef定义(见stdio.h)

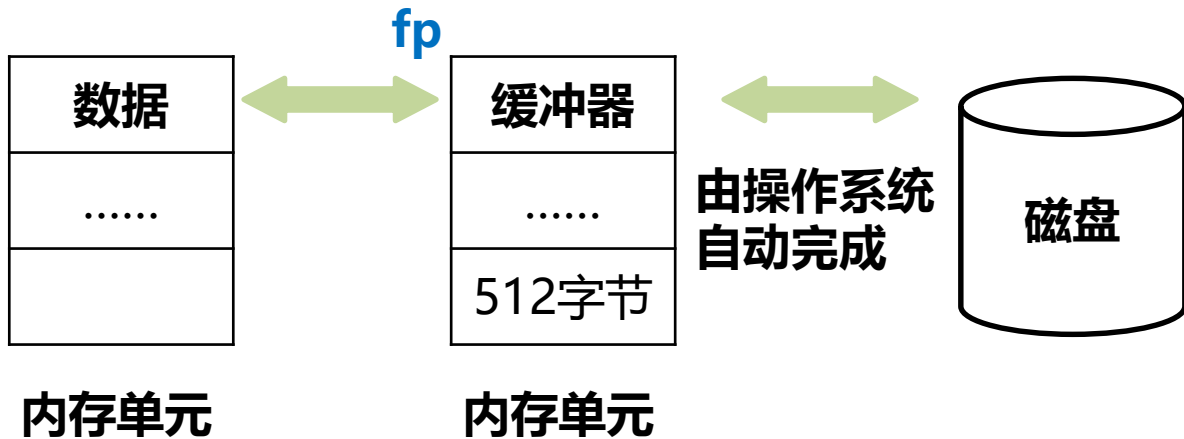
```
typedef struct{
    short          level;          /* 缓冲区使用量 */
    unsigned       flags;          /* 文件状态标志 */
    char           fd;             /* 文件描述符 */
    short          bsize;          /* 缓冲区大小 */
    unsigned char  *buffer;        /* 文件缓冲区的首地址 */
    unsigned char  *curp;          /* 指向文件缓冲区的工作指针 */
    unsigned char  hold;           /* 其他信息 */
    unsigned       istemp;
    short          token;
} FILE;
```

文件类型指针

FILE *fp;

如何使fp与具体文件挂钩？

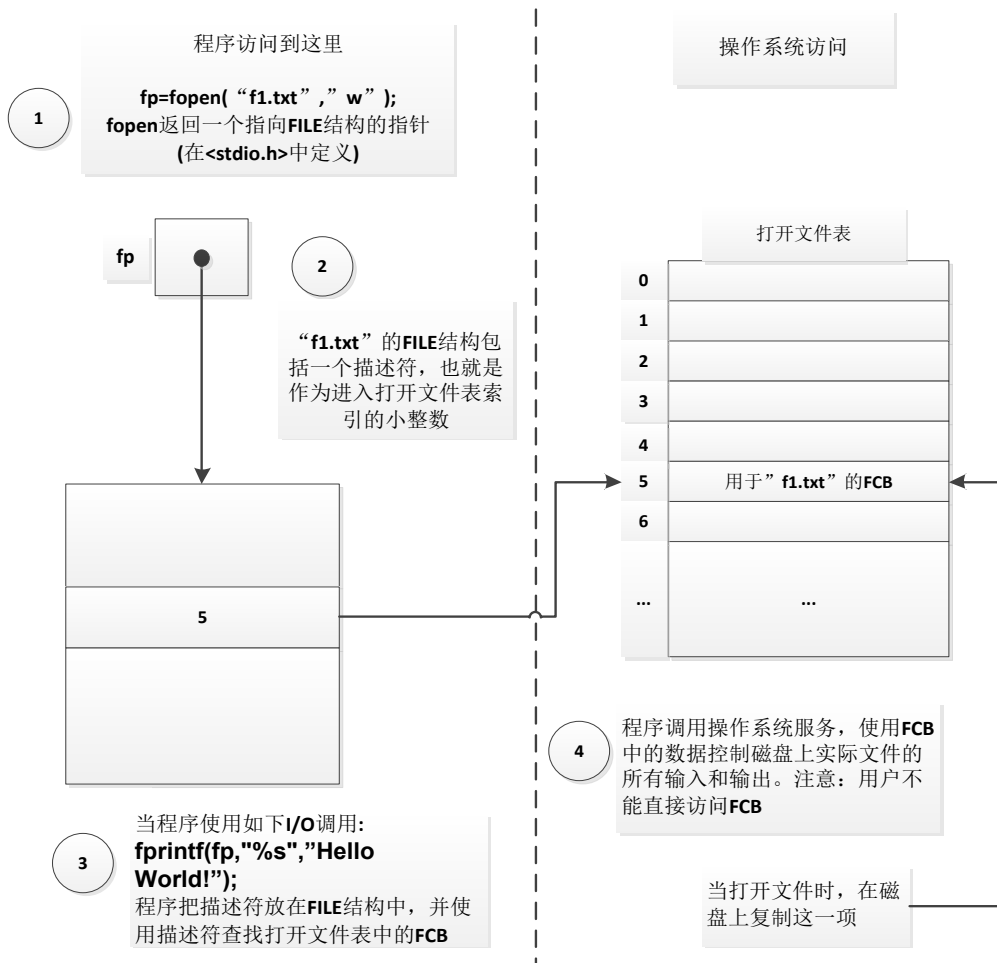
□ 指向文件缓冲区，通过移动指针实现对文件的操作



□ 同时使用多个文件时，每个文件都有缓冲区，用不同的文件指针分别指示。

文件控制块FCB

- ❑ 文件控制块FCB(File Control Block)
- ❑ OS中对文件的操作控制是通过FCB，处理的是FCB列表
- ❑ 一个文件对应一个FCB
- ❑ 文件缓冲区由程序中fopen语句动态创建
- ❑ 打开文件时，FCB的内容信息被复制到文件缓冲区保存
- ❑ 用文件指针指向文件缓冲区实现对文件数据的访问



文件处理步骤

□ 四个步骤：

- 定义文件指针

- 打开文件：文件指针指向磁盘文件缓冲区

- 文件处理：文件读写操作

- 关闭文件

用户信息加密和校验

□ 例12-2 为了保障系统安全，通常采用用户账号和密码登录系统。系统用户信息存放在一个文件中，系统账号名和密码由若干字母与数字字符构成，因安全需要文件中的密码不能是明文，必须要经过加密处理。请编程实现：输入5个用户信息（包含账号名和密码）并写入f12-2.dat。要求文件中每个用户信息占一行，账号名和加密过的密码之间用一个空格分隔。**密码加密算法**：对每个字符ASCII码的低四位求反，高四位保持不变（即将其与15进行异或）。

程序解析

```
#include <stdio.h>
#include <string.h>
struct sysuser{ /*定义系统用户帐号信息结构*/
    char username[20];
    char password[8];
};
int main(void)
{
    FILE *fp; /*1.定义文件指针*/
    int i;
    void encrypt(char *pwd);
    struct sysuser su;
    if((fp=fopen("f12-2.txt","w")) == NULL){
        printf("File open error!\n");
        exit(0);
    }
    for(i=1;i<=5;i++){
        printf("Enter %i th sysuser(name password):",i);
        scanf("%s%s",su.username,su.password);
        encrypt(su.password);
        fprintf(fp,"%s %s\n",su.username,su.password);
    }
    if(fclose(fp)){
        printf("Can not close the file!\n");
        exit(0);
    }
    return 0;
}
```

```
/*加密算法*/
void encrypt(char *pwd)
{
    int i;
    /*与15 (二进制码是00001111) 异或, 实现低四位取反, 高四位保持不变*/
    for(i=0;i<strlen(pwd);i++)
        pwd[i] = pwd[i] ^ 15;
}

/*2.打开文件, 进行写入操作*/

/*3. 将5位用户帐号信息写入文件*/

/*输入用户名和密码 */
/*进行加密处理*/

/*写入文件*/

/*4.关闭文件*/
```

打开文件和关闭文件

```
if((fp = fopen("f12-2.txt", "w")) == NULL){  
    printf("File open error!\n");  
    exit(0);  
}
```

□ fopen("文件名", "文件打开方式")

- 使文件指针与相应文件实体对应起来
- 程序对文件指针进行操作，即fp代表磁盘文件

□ 函数fopen()的返回值

- 执行成功，则返回包含文件缓冲区等信息的FILE型地址，赋值给文件指针fp
- 不成功，则返回一个NULL（空值）

exit(0)：关闭所有打开的文件，并终止程序的执行

参数0表示程序正常结束；非0参数通常表示不正常的程序结束

文件打开方式

```
fp = fopen("f12-2.txt", "w")
```

□ 文件打开方式参数表

文 本 文 件 (ASCII)		二 进 制 文 件(Binary)	
使用方式	含 义	使用方式	含 义
“ r ”	打开只读文件	“ rb ”	打开只读文件
“ w ”	建立只写新文件	“ wb ”	建立只写新文件
“ a ”	打开添加写文件	“ ab ”	打开添加写文件
“ r+ ”	打开读/写文件	“ rb+ ”	打开读/写文件
“ w +”	建立读/写新文件	“ wb+ ”	建立读/写新文件
“ a +”	打开读/写文件	“ ab+ ”	打开读/写文件

文件读写与打开方式

if 读文件

指定的文件必须存在，否则出错；

if 写文件(指定的文件可以存在，也可以不存在)

if 以 "w" 方式写

if 该文件已经存在

原文件将被删去重新建立；

else

按指定的名字新建一个文件；

else if 以 "a" 方式写

if 该文件已经存在

写入的数据将被添加到指定文件原有数据的后面，不会删去原来的内容；

else

按指定的名字新建一个文件（与 "w" 相同）；

if 文件同时读和写

使用 "r+"、"w+" 或 "a+" 打开文件

关闭文件

```
if(fclose(fp)){  
    printf("Cannot close the file!\n");  
    exit(0);  
}
```

□ fclose(文件指针)

- 把缓冲区中的数据写入磁盘扇区，确保写文件的正常完成
- 释放文件缓冲区单元和FILE结构体，使文件指针与具体文件脱钩。

□ 函数fclose()的返回值

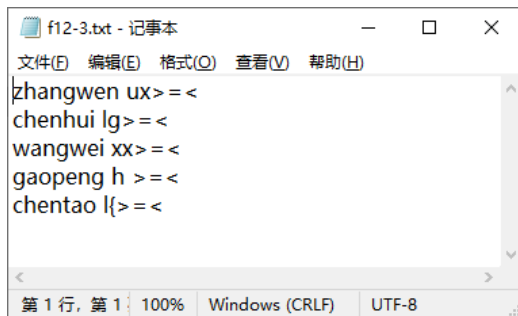
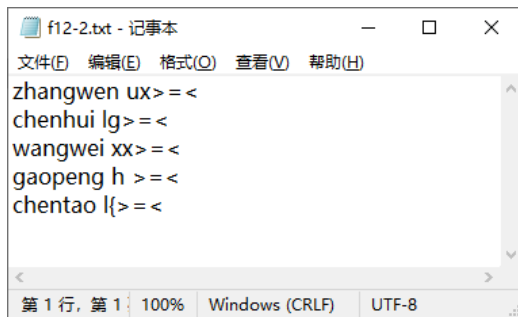
- 返回0：正常关闭文件
- 返回非0：无法正常关闭文件

文件读写

- 例12-3 赋值用户信息文件。将例12-2的用户信息文件f12-2.txt文件备份一份，取名为文件f12-3.txt。
- 说明：运行程序前将文件f12-2.txt与源程序放在同一目录下。

例12-3 源程序

```
#include <stdio.h>
#include <process.h>
int main(void)
{
    FILE *fp1,*fp2;
    char ch;
    if((fp1=fopen("f12-2.txt","r"))==NULL){
        printf("File open error!\n");
        exit(0);
    }
    if((fp2=fopen("f12-3.txt","w"))==NULL){
        printf("File open error!\n");
        exit(0);
    }
    while(!feof(fp1)){
        ch=fgetc(fp1);
        if(ch != EOF) fputc(ch,fp2);
    }
    if(fclose(fp1)){
        printf("Can not close the file!\n");
        exit(0);
    }
    if(fclose(fp2)){
        printf("Can not close the file!\n");
        exit(0);
    }
    return 0;
}
```



打开多个文件

```
if((fp1=fopen("f12-2.txt","r"))==NULL){  
    printf("File open error!\n");  
    exit(0);  
}  
if((fp2=fopen("f12-3.txt","w"))==NULL){  
    printf("File open error!\n");  
    exit(0);  
}
```

□ C语言允许同时打开多个文件

□ 不同的文件对应不同的文件指针

□ 不允许同一个文件在关闭前再次打开

文件读写函数

- 字符读写函数: `fgetc()` / `fputc()`
- 字符串读写函数: `fgets()` / `fputs()`
- 格式化读写函数: `fscanf()` / `fprintf()`
- 二进制读写函数: `fread()` / `fwrite()`
- 其它相关函数:
 - 检测文件结尾函数 `feof()`
 - 检测文件读写出错函数 `ferror()`
 - 清除末尾标志和出错标志函数 `clearerr()`
 - 文件定位的函数 `fseek()`、`rewind()`、`ftell()`

字符读写函数fgetc()和fputc()

□ 函数fgetc()

□ `ch = fgetc(fp);`

从fp所指示的磁盘文件上读入一个字符到ch

□ 区分键盘字符输入函数getchar()

□ 函数fputc()

□ `fputc(ch, fp)`

把一个字符ch写到fp所指示的磁盘文件上

□ 返回值

□ -1(EOF): 写文件失败

□ ch: 写文件成功

```
while(!feof(fp1)){  
    ch = fgetc(fp1);  
    if(ch != EOF) fputc(c, fp2);  
}
```

字符串读写函数fgets()和fputs()

□ 函数fputs()

□ fputs(s, fp);

向指定的文本文件写入一个字符串

□ s: 要写入的字符串, 结束符'\0'不写入文件

□ 函数返回值

□ 执行成功, 函数返回所写的最后一个字符

□ 否则, 函数返回EOF

字符串读写函数fgets()和fputs()

□ 函数fgets()

□ fgets(s, n, fp);

从文本文件中读取字符串

- s: 可以是字符数组名或字符指针; n: 指定读入的字符个数; fp: 文件指针
- 函数被调用时, 最多读取n-1个字符, 并将读入的字符串存入s所指向内存地址开始的n-1个连续的内存单元中。当函数读取的字符达到指定的个数或接收到换行符, 或接收到文件结束标志EOF时, 将在读取的字符后面自动添加一个'\0'字符; 若有换行符, 则将换行符保留(换行符在'\0'之前); 若有EOF, 则不保留
- 函数返回值
 - 执行成功, 函数返回读取的字符串
 - 否则, 则返回空指针, 这时, s的内容不确定

例12-4

- 例12-2的f12-2.txt文件保存着系统用户信息，编写一个函数checkUserValid()用于登录系统时校验用户的合法性。检查方法是：
 - 在程序运行时输入用户名和密码，然后在用户文件中查找该用户信息，如果用户名与密码在文件中找到，则表示用户合法，返回1，否则返回0。
 - 程序运行时，输入一个用户名和密码，调用checkUserValid()函数，如果返回1，则提示"Valid user!"，否则输出"Invalid user!"。
 - 提示：合法性检查的规则。由于文件中的用户名和密码按行存取，把一行看作一个字符串s1，将输入的用户名和密码加密后生成另一个字符串s2，然后通过比较s1和s2，来确定文件中是否存在用户。

格式化文件读写fscanf()和fprintf()

□ 指定格式的输入输出函数:

□ fscanf(文件指针, 格式字符串, 输入表);

□ fprintf(文件指针, 格式字符串, 输出表);

```
FILE *fp; int n; float x;
```

```
fp = fopen("a.txt", "r");
```

```
fscanf(fp, "%d%f", &n, &x);
```

从文件a.txt分别读入整型数到变量n、浮点数到变量x

```
fp = fopen("b.txt", "w");
```

```
fprintf(fp, "%d%f", n, x);
```

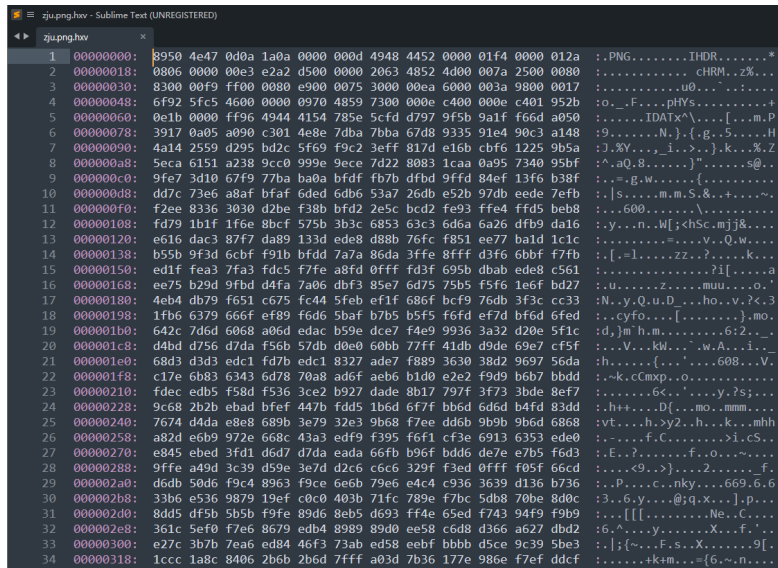
将变量n和x的数值写入文件b.txt

二进制文件

□ 二进制文件用文本方式显示会显示乱码，因为其本身不表示字符

□ 查看二进制字节码可以使用：

□ sublime、notepad++等编辑器（需要安装相应的插件），或是编写程序输出



Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000030	83	00	00	f9	ff	00	00	80	e9	00	00	75	30	00	00	ea	?.?..€?.u0..?□□□
00000040	60	00	00	3a	98	00	00	17	6f	92	5f	c5	46	00	00	00	`.....o扣臚...[
00000050	09	70	48	59	73	00	00	0e	c4	00	00	0e	c4	01	95	2b	.pHYs...?..?□□□
00000060	0e	1b	00	00	ff	96	49	44	41	54	78	5e	5c	fd	d7	97 眇DATx^\ ？
00000070	9f	5b	9a	1f	6f	6d	a0	50	39	17	0a	05	a0	90	c3	01	焱?鯉.P9.....?□□□
00000080	4e	8e	7d	ba	7b	ba	67	d8	93	35	91	a4	90	c3	a1	48	N筊筋筭負5或.漫HC
00000090	4a	14	25	59	d2	95	bd	2c	5f	69	f9	c2	3e	ff	81	7d	J.¥Y視?_i > .}[
000000a0	e1	6b	cb	f6	12	25	9b	5a	5e	ca	61	51	a2	38	9c	c0	營琐.％迥+蔭Q?澀□□
000000b0	99	9e	9e	ce	7d	22	80	83	1c	aa	0a	95	73	40	95	bf	襪坂)"e??眇@囁□□□
000000c0	9f	e7	3d	10	67	f9	77	ba	ba	0a	bf	df	fb	7b	df	bd	煊=.g鵠汉.窳麟吃□
000000d0	9f	fd	84	ef	13	f6	b3	8f	dd	7c	73	e6	a8	af	bf	af	燒粿.訛.軫s姉 ?[
000000e0	6d	ed	6d	b6	53	a7	26	db	e5	2b	97	db	ee	d2	7e	fb	m鞘機?垢+棹鉅~?□□
000000f0	f2	ee	83	36	30	30	d2	be	f3	8b	bf	d2	2e	5c	bc	d2	蝶?00揖髻忌.~家□□
00000100	fe	93	ff	e4	ff	d5	be	b8	fd	79	1b	1f	1f	6e	8b	cf	鉶 ?站庚y...n嬭□
00000110	57	5b	3b	3c	68	53	63	c3	6d	6a	6a	26	df	b9	da	16	W[;<hSc胛jj&Y?□□
00000120	e6	16	da	c3	87	f7	da	89	13	3d	ed	e8	d8	8b	76	fc	?谈圖越.=窳瓩v?□□
00000130	f8	51	ee	77	ba	1d	1c	1c	b5	5b	9f	3d	6c	bf	f9	1b	鳴顆?..礫?1岢.□□□
00000140	bf	dd	7a	7a	86	da	3f	fe	8f	ff	d3	f6	6b	bf	f7	fb	枯zz囉?? 遇k亏?[
00000150	ed	1f	fe	a3	7f	da	3f	cd	c5	f7	fe	a8	fd	0f	ff	fd	3f ? .} 还 ? ?□□□

数据块读写fread()和fwrite()

□ fread(buffer, size, count, fp);

从二进制文件中读入一个数据块到变量

□ fwrite(buffer, size, count, fp);

向二进制文件中写入一个数据块

□ 参数:

- **buffer**: 指针, 表示存放数据的首地址
- **size**: 数据块的字节数
- **count**: 要读写的数据块块数
- **fp**: 文件指针

□ 返回值:

- 返回成功读取 (写入) 的数据块个数, 如果返回的个数与传入的count参数不同, 则可能发生了错误 (对于fread函数来说也可能是到达了文件末尾)

例：实现copy命令

□ 用二进制文件读写方法实现文件copy命令。

```
#include <stdio.h>
#include <stdlib.h>

#define BUFFER_SIZE 1024

int main(int argc, char *argv[])
{
    FILE *fin, *fout;
    int rc; //read count
    unsigned char buf[BUFFER_SIZE];

    if(argc < 3){
        printf("usage: %s %s\n", argv[0], "srcfile targetfile");
        exit(1);
    }
    if((fin = fopen(argv[1], "rb")) == NULL){
        printf("failed to open source file %s\n", argv[1]);
        exit(1);
    }
    if((fout = fopen(argv[2], "wb")) == NULL){
        printf("failed to open target file %s\n", argv[2]);
        fclose(fin);
        exit(1);
    }
}
```

```
while((rc = fread(buf, sizeof(unsigned char), BUFFER_SIZE, fin)) != 0)
    fwrite(buf, sizeof(unsigned char), rc, fout);

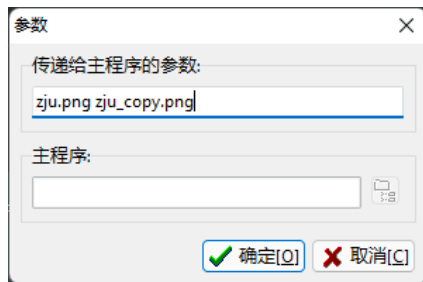
fclose(fin), fclose(fout);

return 0;
}
```

方法一：源文件编译后打开命令行窗口执行下面的命令：

```
copy.exe zju.png zju_copy.png
```

方法二：Dev C++，运行->参数，设置传递给程序的参数：



例：识别PNG格式文件

□ PNG是一种适用于网络传播的无损图片格式。其规定文件开头的8字节的PNG文件署名为0x89504e470d0a1a0a。试编写程序验证PNG文件的文件头署名。

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main()
{
    unsigned char buffer[8];
    FILE *fp;

    if((fp = fopen("zju.png", "rb")) == NULL){
        printf("File open error!\n");
        exit(1);
    }
```

```
    fread(buffer, 1, 8, fp);
    for(int i = 0; i < 8; i++)
        printf("%02hhx ", buffer[i]);
```

```
    fclose(fp);
    return 0;
}
```

zju.png



```
00000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452 0000 01f4 0000 012a :.PNG.....IHDR.....*
00000018: 0800 0000 0000 e2a2 d500 0000 2063 4852 4d00 007a 2500 0080 :.....cHRM..z%...
00000030: 8300 00f9 ff00 0080 e900 0075 3000 00ea 6000 003a 9800 0017 :.....u0...`.....
00000048: 6f92 5fc5 4600 0000 0970 4859 7300 000e c400 000e c401 952b :o...F....pHYs.....+
00000060: 0e1b 0000 ff96 4944 4154 785e 5cfd d797 9f5b 9a1f f66d a050 :.....IDATx^.....[...m.P
00000078: 3917 0a05 a090 c301 4e8e 7dba 7bba 67d8 9335 91e4 90c3 a148 :9.....N.}.{g..5....H
00000090: 4a14 2559 d295 bd2c 5f69 f9c2 3eff 817d e16b cbf6 1225 9b5a :J.%Y..._i...>..}.k...%Z
000000a8: 5eca 6151 a238 9cc0 999e 9ece 7d22 8083 1caa 0a95 7340 95bf :^..aQ.8.....}".....s@..
000000c0: 9fe7 3d10 67f9 77ba ba0a bfdf fb7b dfbd 9ffd 84ef 13f6 b38f :...=.g.W.....{.....
```

例：读取PNG图像的宽和高

- PNG图像文件将图像的宽和高记录在文件起始16字节处，宽和高分别占用4个字节。试编写程序输出某个PNG图像的宽和高。

zju.png (500 x 298)



$$0000\ 01f4 = 1 \times 16^2 + 15 \times 16 + 4 = 500$$

$$0000\ 012a = 1 \times 16^2 + 2 \times 16 + 10 = 298$$

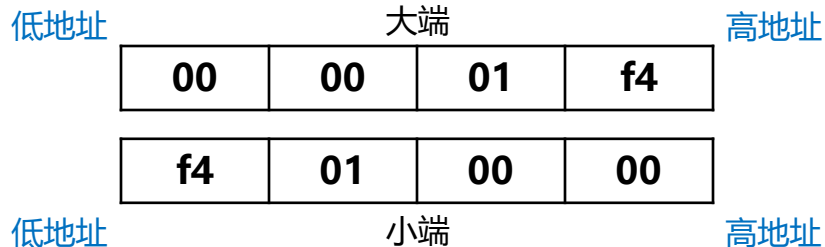
```
00000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452 0000 01f4 0000 012a :.PNG.....IHDR.....*
00000018: 0806 0000 00e3 e2a2 d500 0000 2063 4852 4d00 007a 2500 0080 :.....cHRM..z%..
00000030: 8300 00f9 ff00 0080 e900 0075 3000 00ea 6000 003a 9800 0017 :.....u0....:....
00000048: 6f92 5fc5 4600 0000 0970 4859 7300 000e c400 000e c401 952b :o_.F....pHYs.....+
00000060: 0e1b 0000 ff96 4944 4154 785e 5cfd d797 9f5b 9a1f f66d a050 :.....IDATx^\....[...m.P
00000078: 3917 0a05 a090 c301 4e8e 7dba 7bba 67d8 9335 91e4 90c3 a148 :9.....N.}.{.g..5....H
00000090: 4a14 2559 d295 bd2c 5f69 f9c2 3eff 817d e16b cbf6 1225 9b5a :J.%Y...,_i...>..}.k...%Z
000000a8: 5eca 6151 a238 9cc0 999e 9ece 7d22 8083 1caa 0a95 7340 95bf :^,aQ.8.....}".....s@..
000000c0: 9fe7 3d10 67f9 77ba ba0a bfdf fb7b dfbd 9ffd 84ef 13f6 b38f :...=.g.w.....{.....
```

源程序：读取PNG图像的宽和高

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

// 大端存储转小端
uint32_t be2le(uint32_t be)
{
    return (
        ((be >> 24) & 0xff) |
        ((be >> 8) & 0xff00) |
        ((be << 8) & 0xff0000) |
        ((be << 24) & 0xff000000)
    );
}
```

对于同样的整数500，大小端模式下其字节码存储顺序不同：



```
int main()
{
    uint8_t width[4], height[4];
    FILE *fp;

    if((fp = fopen("zju.png", "rb")) == NULL){
        printf("File open error!\n");
        exit(1);
    }

    // png图像的宽和高位于文件起始16字节处,
    // 分别占4个字节
    fseek(fp, 16L, SEEK_SET);
    fread(width, 1, 4, fp);
    fread(height, 1, 4, fp);

    // png文件中存放宽和高的数据是大端的顺序
    // 需要在内存中转成小端才能在x86等小端处理器上正常打印整数
    printf("png image width: %d, height: %d",
        be2le(*(uint32_t*)width),
        be2le(*(uint32_t*)height));

    fclose(fp);

    return 0;
}
```

其它相关函数

□ 函数feof()

□ feof(fp);

判断fp指针是否已经到达文件末尾

□ 返回值

□ 1: 到文件结束位置

□ 0: 文件未结束

□ 函数rewind()

□ rewind(fp)

定义文件指针，使文件指针指向读写文件的首地址，即打开文件时文件指针所指向的位置。

其它相关函数

□ 函数fseek(): 控制指针移动

□ `fseek(fp, offset, from);`

□ offset: 移动偏移量, long型

□ from: 起始位置, 文件首部、当前位置和文件尾部分别对应0, 1, 2, 或常量SEEK_SET、SEEK_CUR、SEEK_END。

□ 例如:

□ `fseek(fp, 20L, 0)`: 将文件位置指针移动到从文件首20字节处

□ `fseek(fp, -20L, SEEK_END)`: 将文件位置指针移动到从文件尾部前20字节处

□ 函数ftell(): 获取当前文件指针的位置 (相对于开头的字节数)

□ `ftell(fp);`

□ 函数出错时, 返回-1L

其它相关函数

□ 函数ferror()

□ `ferror(fp);`

- 检查文件在用各种输入输出函数进行读写时是否出错，若返回值为0，表示未出错，否则表示出错

□ 函数clearerr()

□ `clearerr(fp);`

- 清空出错标志和文件结束标志，使它们为0

谢 谢

位段

- 为了节省存储空间，并使处理简便，C语言提供了又一种使用结构的方法——位段（也称位域）。所谓位段就是以位为单位定义长度的结构体类型中的成员。每个位段都有一个位段名，允许在程序中引用位段中的数据进行操作。这样就可以将若干个信息紧缩存放，仅用一个或几个字节来表示。

- 定义：

```
struct <结构体名>
```

```
{ <位段表列>
```

```
};
```

```
struct <结构体名> <结构变量表列>;
```

```
struct <结构体名>
```

```
{ <位段表列>
```

```
}<结构变量表列>;
```

```
struct
```

```
{ <位段表列>
```

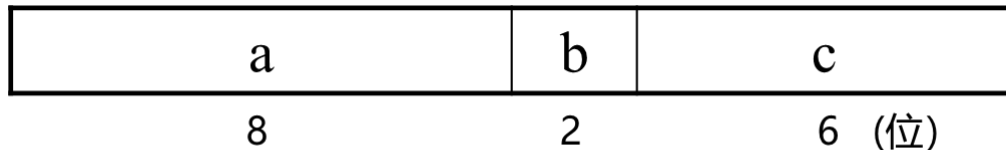
```
}<结构变量表列>;
```

- 在这三种形式中，位段表列的形式均为：
- <类型说明符> <位段名>：<位段长度>;

位段

例如: struct bs
{ unsigned a:8;
 unsigned b:2;
 unsigned c:6;
}data;

表示data为struct bs型结构变量（也称位段变量），共占两个字节。其中位段a占8位，位段b占2位，位段c占6位，如图所示。



data位段变量内存示意图

位段

- 位段在本质上就是一种结构体类型中的成员，只不过是按二进制位分配的。因此，位段中数据的引用可按结构变量成员的访问方法进行，形式为：
- `<结构变量>.<位段名>`
- 如：`data.a=2`;表示给位段变量data的位段a赋值为2。

位段：说明

- ❑ 一个位段必须存储在同一个存储单元中，不能跨跃两个单元。如一个单元所剩空间不够存放下一位段时，应从下一单元起存放该位段。
- ❑ 可以有意使某位段从下一单元开始存放。
- ❑ 位段中的数据不能超过其允许的最大值范围，位段的长度不能大于机器字长。例如，定义位段的长度为2位，则其中存放的最大数为3。
- ❑ 位段可以无名位段，无名位段是不能使用的，它只用来作填充或调整位置。
- ❑ 不能定义位段数组，也不能定义返回值为位段的函数。
- ❑ 位段允许在数据表达式中引用，它会被系统自动地转换为整型数。

位段【例】

□ 位段变量和指向位段的指针变量的使用

```
#include "stdio.h"
main()
{
    struct bs
    {   unsigned a:1;
        unsigned b:3;
        unsigned c:4;
    } bit,*pbit;
    bit.a=1;           /*分别给三个位段赋值*/
    bit.b=7;
    bit.c=15;
    printf("%d,%d,%d\n",bit.a,bit.b,bit.c); /*以整型量格式输出三个位段的内容*/
    pbit=&bit;          /*把位段变量bit的地址送给指针变量pbit*/
    pbit->a=0;          /*用指针方式给位段a重新赋值，赋为0*/
}
```

位段【例】

```
pbit->b&=3;
```

```
pbit->c|=1;
```

```
printf("%d,%d,%d\n",pbit->a,pbit->b,pbit->c);
```

/*用指针方式输出了这三个位段的值*/

```
}
```

运行结果为：

1, 7, 15

0, 3, 15