

# 第10章 对文件的输入输出（3）



# 复习回顾

## ➤ 上次课的内容：

◆ 文件顺序读写

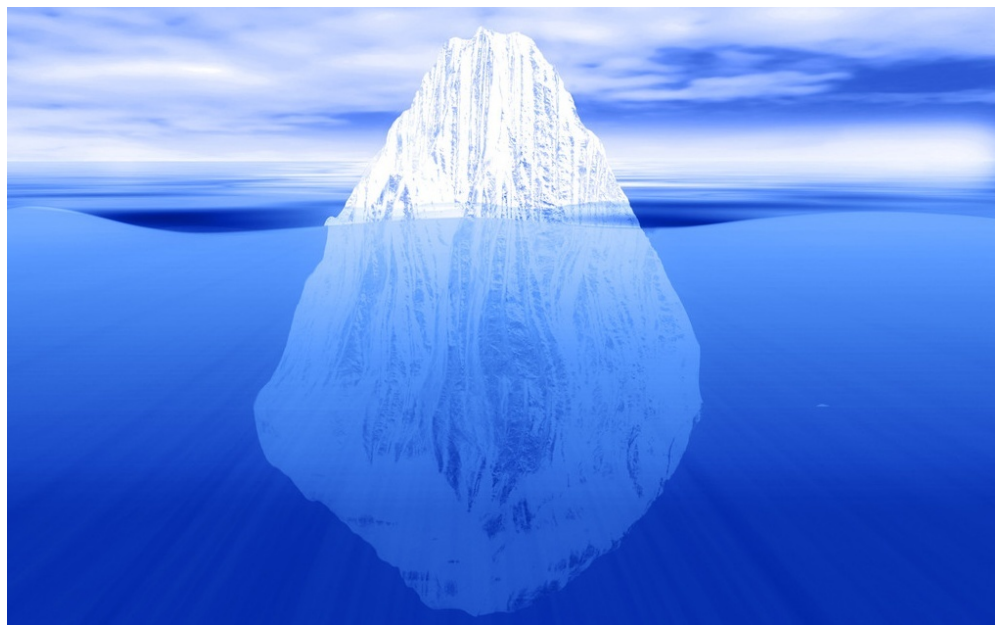
◆ C语言中的流

◆ 文件的缓冲区

◆ 文件操作举例

◆ 即将开始期末复习，但是我们真的把C语言学完了吗？

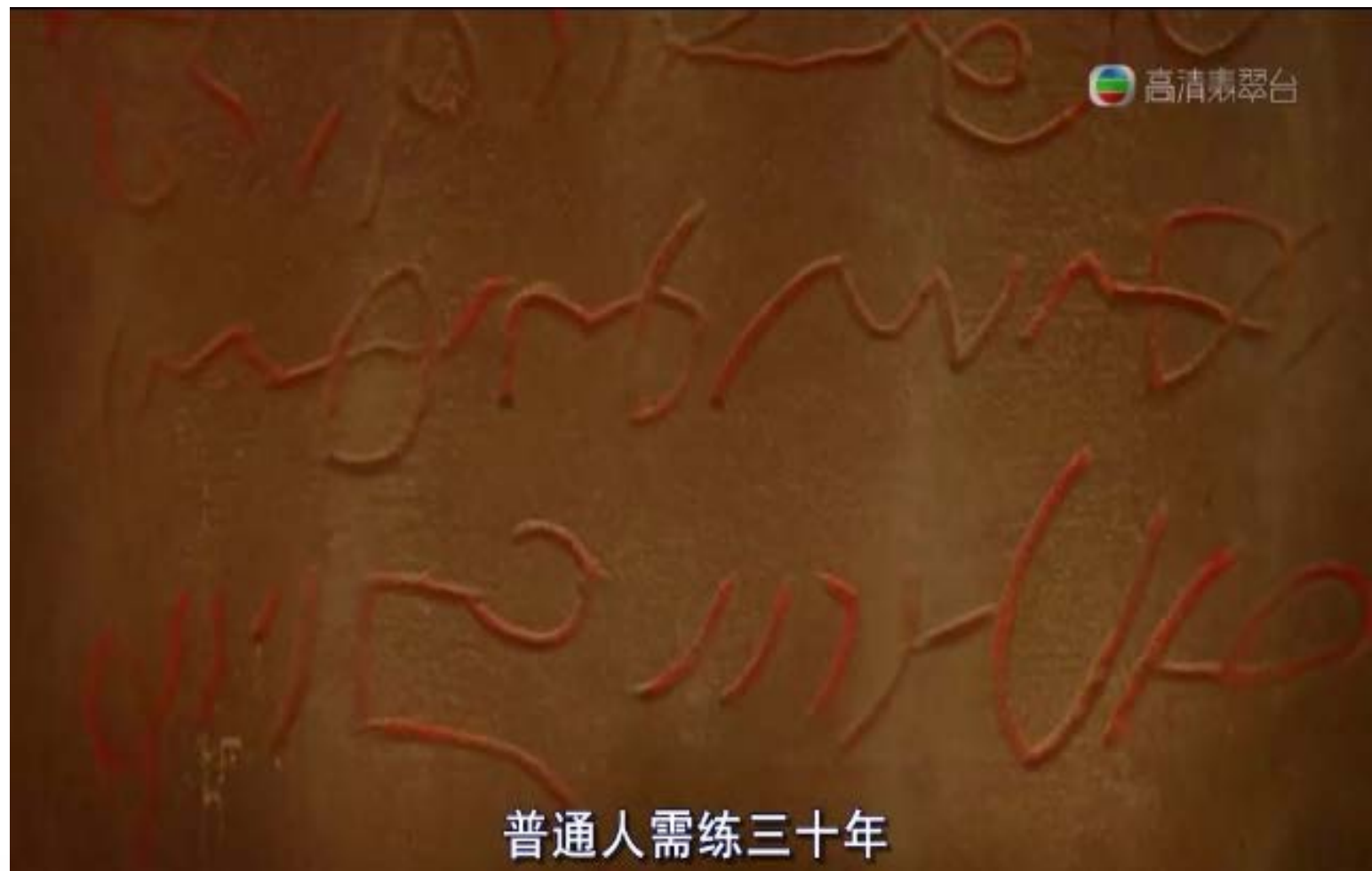
◆ 所学冰山一角而已。那么，**学这门课的真正意义是什么？**



# 当张无忌想练乾坤大挪移心法



# 当张无忌想练乾坤大挪移心法



# 当张无忌想练乾坤大挪移心法



# 当张无忌想练乾坤大挪移心法





# 学习这门课是为了打通任督二脉

字符集/标识符/关键字/语句

常量与变量/数据类型

数组/结构体

地址/指针

讲还没完

文件

.....

基本结构：  
顺序/分支/循环

函数

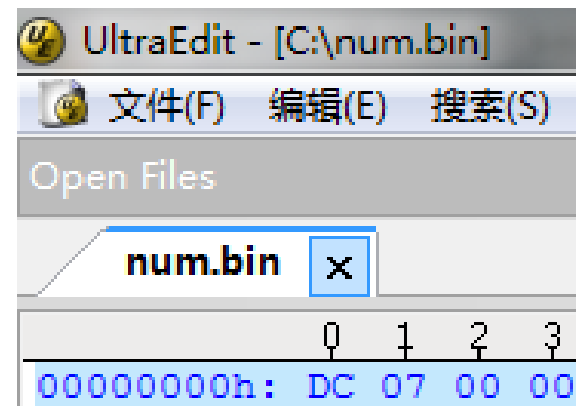
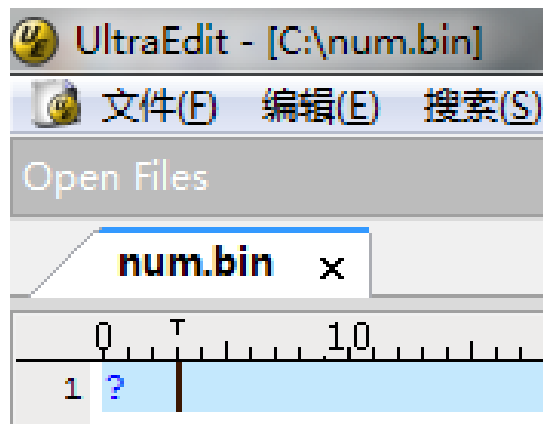
算法思想：穷举  
/递归/排序.....

知其然  
且知其所以然  
学其他编程语言就很快

# 创建二进制文件并写入一个整数

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

```
3. int main()
4. {
5.     FILE * fp;
6.     int i=2012;
```



```
7.     fp = fopen("c:\\num.bin", "wb"); //“文本只写” 方式打开文件
8.     if (fp==NULL)
9.         exit(0);
```

```
10.     fwrite(&i, sizeof(int), 1, fp);
11.     fclose(fp); //关闭文件
12.     return 0;
13. }
```

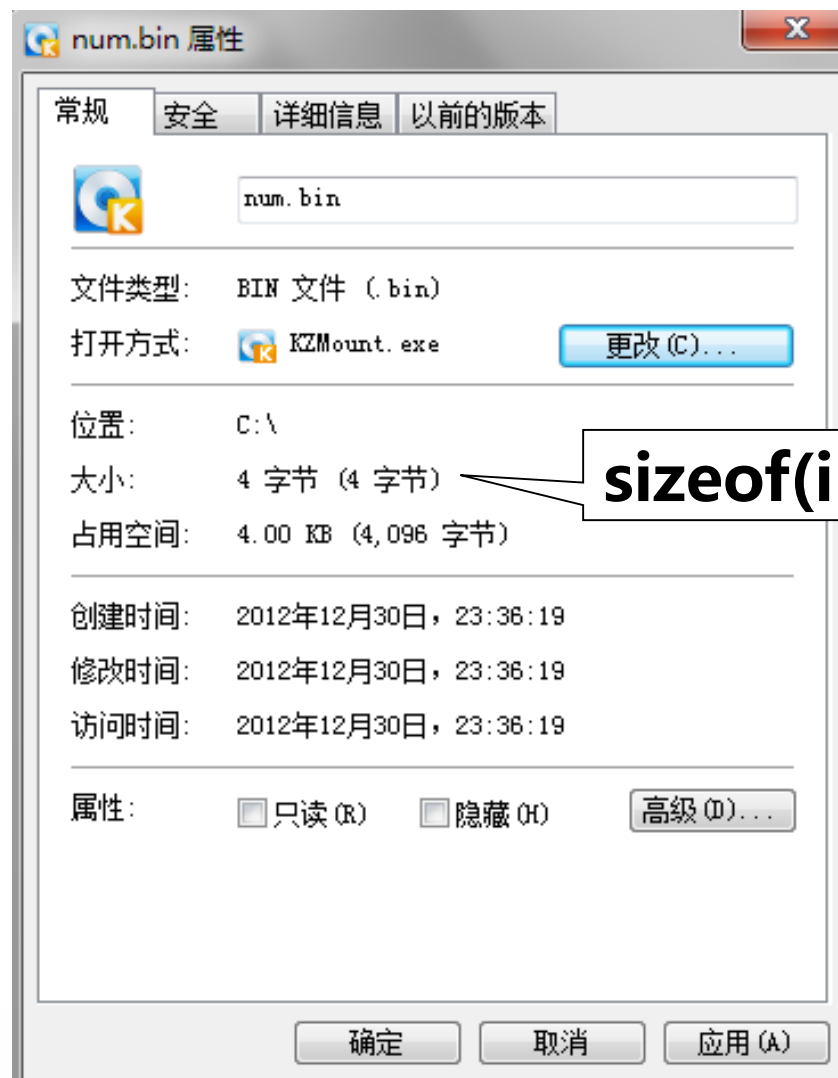
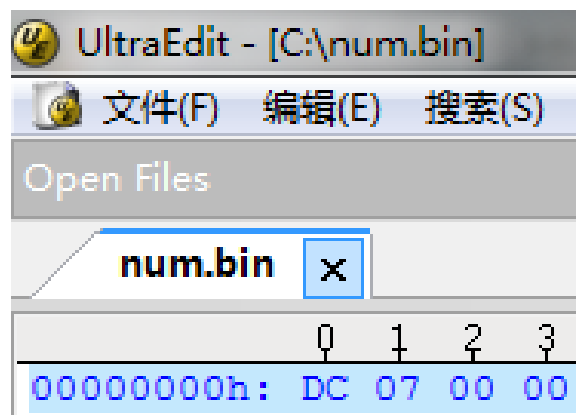
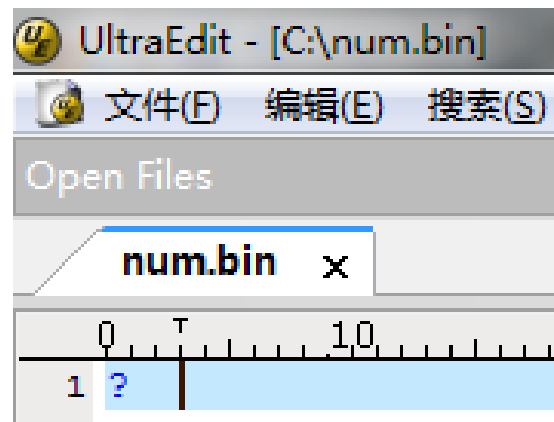
$$2012 = 7 \times 16^2 + 13 \times 16 + 12$$

↑  
D

↑  
C



# 二进制文件的属性



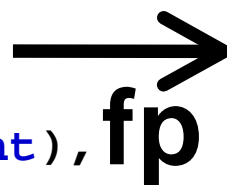
# 写入整数就读取整数

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

```
3. int main()
4. {
5.     FILE * fp;
6.     int i;
```

```
7.     fp = fopen("c:\\num.bin", "rb"); //“文本只读” 方式打开文件
8.     if (fp==NULL)
9.         exit(0);
10.    fread (&i, sizeof(int), fp
11.           1, fp);
12.    printf("%d\n", i);
13.    fclose(fp); //关闭文件
14.    return 0;
15. }
```

```
2012
Press any key to continue
```



DC	07	00	00	EOF
----	----	----	----	-----

fread()后位置标记

# 创建二进制文件并写入一个数组

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

```
3. int main()
```

```
4. {
```

```
5.     FILE * fp;
```

```
6.     float arr[2]={20, 12};
```

```
7.     fp = fopen("c:\\num.bin", "wb"); //“文本只写”方式打开文件
```

```
8.     if (fp==NULL)
```

```
9.         exit(0);
```

```
10.    fwrite(arr, sizeof(float),
```

```
11.           2, fp);
```

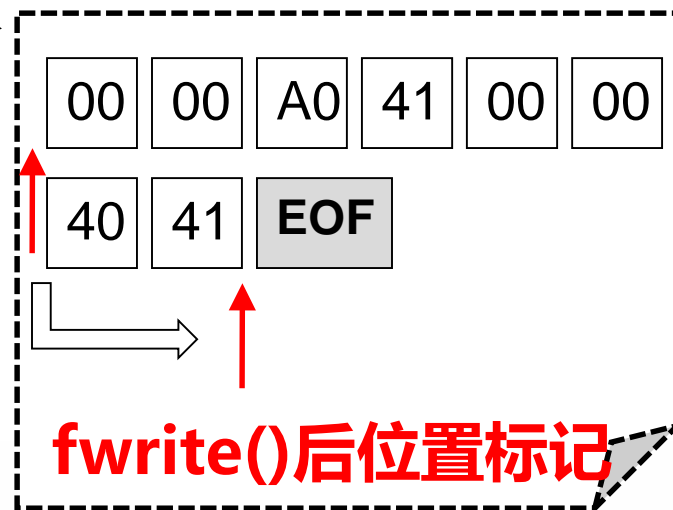
```
12.    fclose(fp); //关闭文件
```

```
13.    return 0;
```

```
14. }
```

num.bin x									
		0	1	2	3	4	5	6	7
00000000h:		00	00	A0	41	00	00	40	41

→  
fp



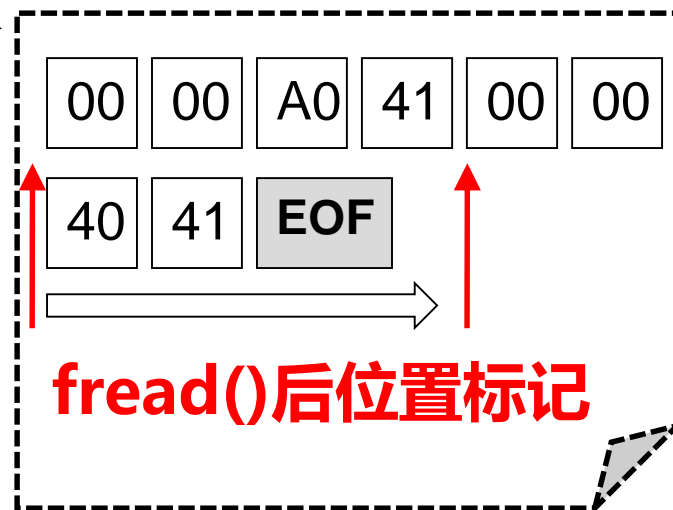
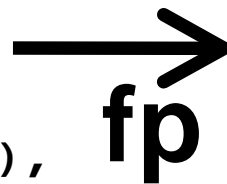
# 读写类型不一致的后果

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

```
3. int main()
4. {
5.     FILE * fp;
6.     int i;
```

```
7.     fp = fopen("c:\\num.bin", "rb"); //“文本只读” 方式打开文件
8.     if (fp==NULL)
9.         exit(0);
10.    fread(&i, sizeof(int),
11.          1, fp);
12.    printf("%d\n", i);
13.    fclose(fp); //关闭文件
14.    return 0;
15. }
```

```
1101004800
Press any key to continue
```



# 通常以写入格式读取二进制文件

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

```
3. int main()
```

```
4. {
```

```
5.     FILE * fp;
```

```
6.     float arr[2];
```

```
7.     int i;
```

```
8.     fp = fopen("c:\\num.bin", "rb"); //“文本只读”方式打开文件
```

```
9.     if (fp==NULL)
```

```
10.         exit(0);
```

```
11.     fread(arr, sizeof(float), fp
```

```
12.         2, fp);
```

```
13.     printf("%f\n", arr[0]);
```

```
14.     printf("%f\n", arr[1]);
```

```
15.     fclose(fp); //关闭文件
```

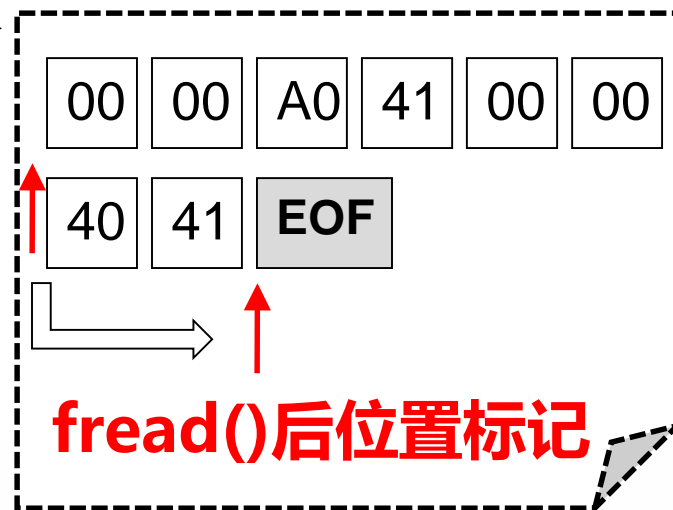
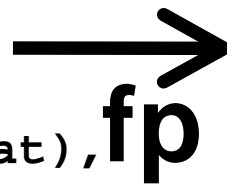
```
16.     return 0;
```

```
17. }
```

```
20.0000000
```

```
12.0000000
```

```
Press any key to continue
```



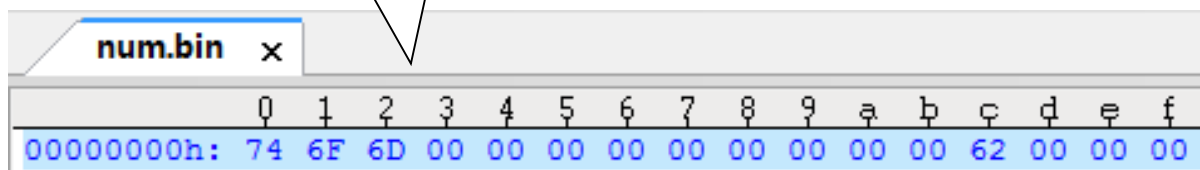
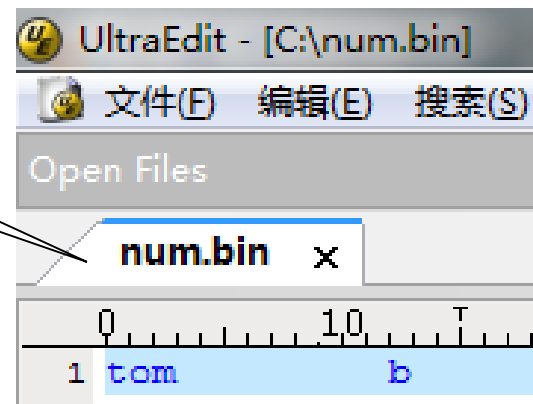
# 创建二进制文件并写入一结构体

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. struct stu_info
4. {
5.     char name[10];
6.     int score;
7. } st = {"tom", 98};
```

```
8. int main()
9. {
10.     FILE * fp;
11.     fp = fopen("c:\\num.bin", "wb"); //“文本只写”方式打开文件
12.     if (fp==NULL)
13.         exit(0);
14.     fwrite(&st, sizeof(struct stu_info), 1, fp);
15.     fclose(fp); //关闭文件
16.     return 0;
17. }
```

文本模式

16进制模式





# 回顾fopen不同字符代表的含义

- **r** ( read ) : 读
- **w** ( write ) : 写
- **a** ( append ) : 追加
- **t** ( text ) : 文本文件 , 可省略不写
- **b** ( binary ) : 二进制文件
- **+** : 读和写

# 关于fopen("f1","r")的说明

(1) 用 **"r" 方式**打开的文件只能用于向计算机输入而不能用作向该文件输出数据，而且该文件应该已经存在，并存有数据，这样程序才能从文件中读数据。

- 不能用 **"r" 方式**打开一个并不存在的文件，否则出错。

# 关于fopen("f1","w")的说明

(2) 用 **"w"** 方式打开的文件只能用于向该文件写数据（即输出文件），而不能用来向计算机输入。

- 如果原来不存在该文件，则在打开文件前新建一个以指定的名字命名的文件。
- 如果原来已存在一个以该文件名命名的文件，则在打开文件前先将该文件删去，然后重新建立一个新文件。

# 关于fopen("f1","a")的说明

(3) 如果希望向文件末尾添加新的数据（不希望删除原有数据），则应该用 **"a" 方式** 打开

- 但此时应保证该文件已存在；否则将得到出错信息。
- 打开文件时，文件读写标记移到文件末尾

# fopen中其他打开方式的说明

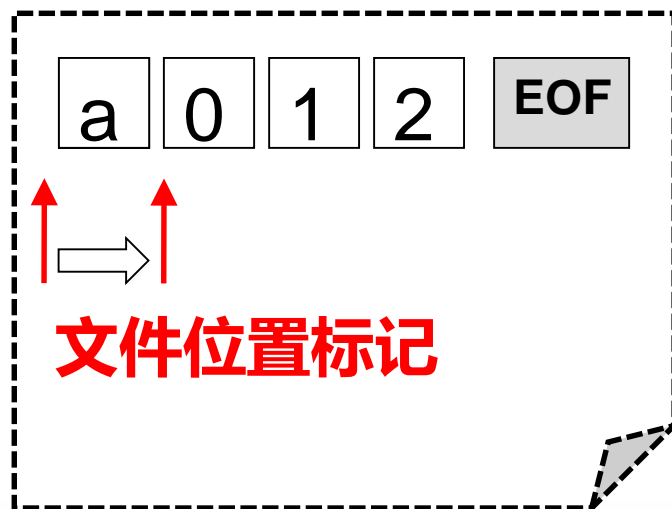
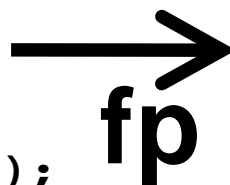
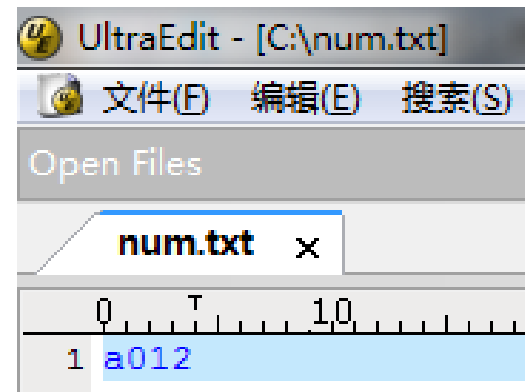
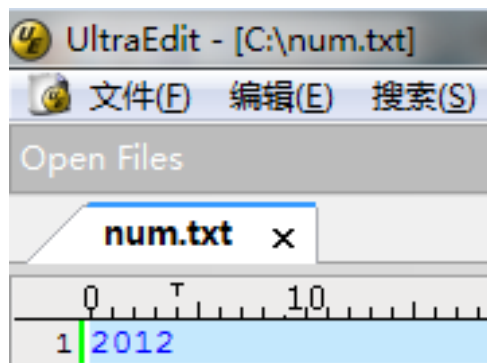
(4) 用**r+**、**w+**、**a+**方式打开的文件既可以用来输入数据，也可以用来输出数据。

- 用**r+**方式时该文件应该已经存在。
- 用**w+**方式则新建立一个文件，先向此文件写数据，然后可以读此文件中的数据。
- 用**a+**方式打开的文件，原来的文件不被删去，文件读写位置标记移到文件末尾，可以添加，也可以读。

# "r+"模式打开文件有什么不同

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

3. int main()
4. {
5.     FILE * fp;
6.     fp = fopen("c:\\num.txt", "r+");
7.     if (fp==NULL)
8.         exit(0);
9.     fprintf(fp, "%c", 'a');
10.    fclose(fp); //关闭文件
11.
12.    return 0;
13. }
```





# 随机读写

- 可以根据读写的需要，人为地移动了文件标记的位置。文件标记可以向前移、向后移，移到文件头或文件尾，然后对该位置进行读写——**随机读写**
- 随机读写可以在**任何位置**写入数据，在任何位置读取数据

# 文件位置标记的定位1

## ➤ 可以强制使文件位置标记指向文件头的位置

◆ 可以用rewind函数实现，函数原型：

```
void rewind(FILE *pfile);
```

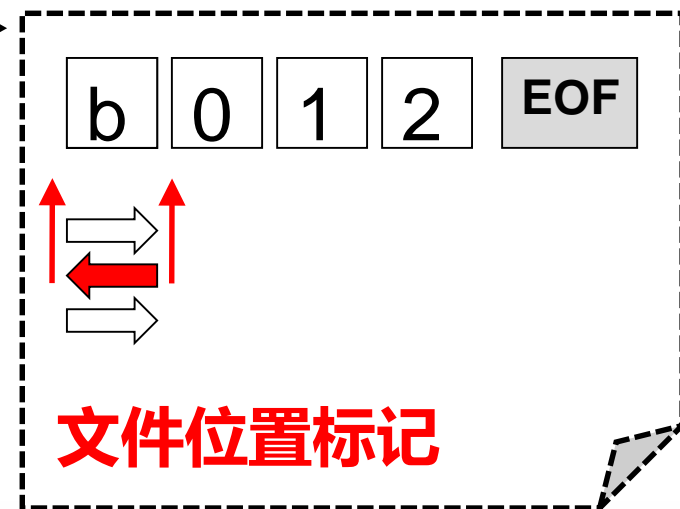
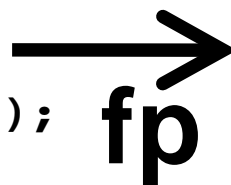
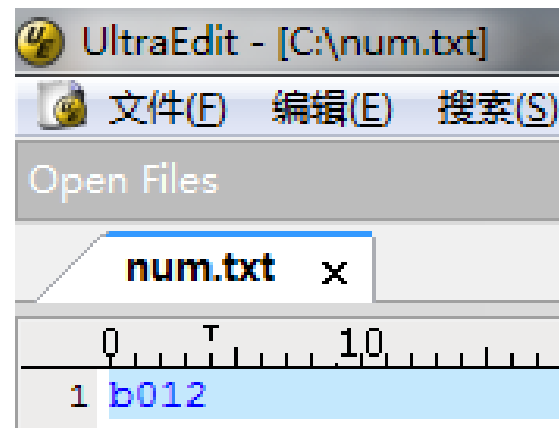
◆ rewind函数的作用是使文件标记**重新返回文件的开头**，此函数没有返回值。

◆ 使用后如果继续向文件中输入数据，将会覆盖原来文件开头的内容。

# 强行把文件位置标记移到文件头

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

3. int main()
4. {
5.     FILE * fp;
6.     fp = fopen("c:\\num.txt", "r+");
7.     if (fp==NULL)
8.         exit(0);
9.     fprintf(fp, "%c", 'a');
10.    rewind(fp);
11.    fputc('b', fp);
12.    fclose(fp); //关闭文件
13.
14.    return 0;
15. }
```



# 使用rewind函数的例子

**例：**有一个磁盘文件，内有一些信息。要求第一次将它的内容显示在屏幕上，第二次把它复制到另一文件上。

## ➤ 解题思路：

- ◆因为在第一次读入完文件内容后，文件标记已指到文件的末尾，如果再接着读数据，就遇到文件结束标志，feof函数的值等于1（真），无法再读数据
- ◆必须在程序中用rewind函数使位置指针返回文件的开头

```
1. #include <stdio.h>
2. int main()
3. {
4.     FILE *fp1,*fp2;
5.     fp1 = fopen("file1.dat", "r");
6.     fp2 = fopen("file2.dat", "w");
7.     while (!feof(fp1))
8.         putchar(getc(fp1));
9.     putchar(10);
10.    rewind(fp1);
11.    while (!feof(fp1))
12.        putc(getc(fp1),fp2);
13.    fclose(fp1);
14.    fclose(fp2);
15.    return 0;
16.}
```

# 补充说明：getc实现字符读入

➤getc的使用形式为：

```
int getc(FILE * pfile);
```

◆调用例如：

```
ch = getc(fp);
```

●表示从pf指针所指向的文件中读取一个字符，如果操作成功则返回该字符，如果文件结束或出错则返回EOF

◆实际上getc并不是一个函数.....而是一个宏



# 补充说明：字符输出 `putc`

➤ `putc`的使用形式为：

```
int putc(int ch, FILE * pfile);
```

◆调用例如：

```
putc(ch, fp);
```

●表示把一个字符`ch`输出到`pf`指针所指向的文件中，若成功则返回该字符的ASCII码，若失败则返回EOF

◆实际上`putc`并不是一个函数.....而是一个宏

## 文件位置标记的定位2

➤ 可以强制使文件标记指向指定的位置

◆ 可以用fseek函数实现，函数原型：

```
int fseek(FILE *pfile, long offset, int base);
```

其中，**pfile**是一个文件类型指针，

**offset**是位移量，表示移动的字节数，

**base**是起始点，表示从何处开始计算位移量

# fseek函数的起始点参数

- C语言规定起始点有三种：**0**代表“文件开始位置”，**1**为“当前位置”，**2**为“文件末尾位置”。
- C标准指定的名字如下表：

起始点	名 字	用数字代表
文件开始位置	SEEK_SET	0
文件当前位置	SEEK_CUR	1
文件末尾位置	SEEK_END	2

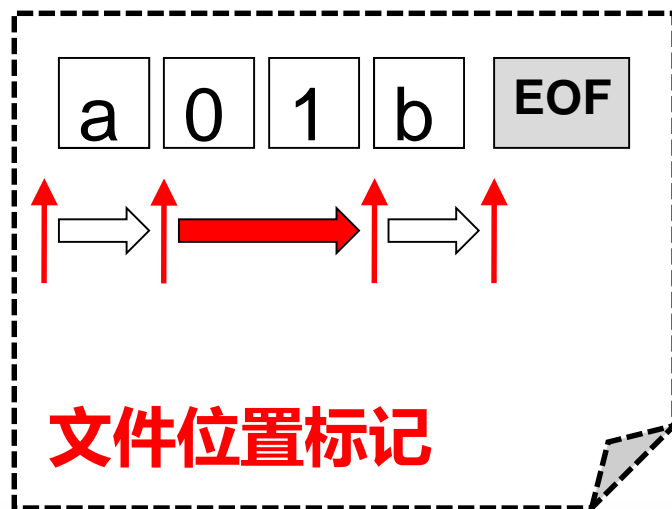
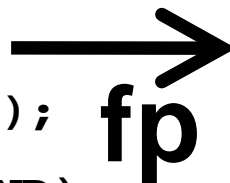
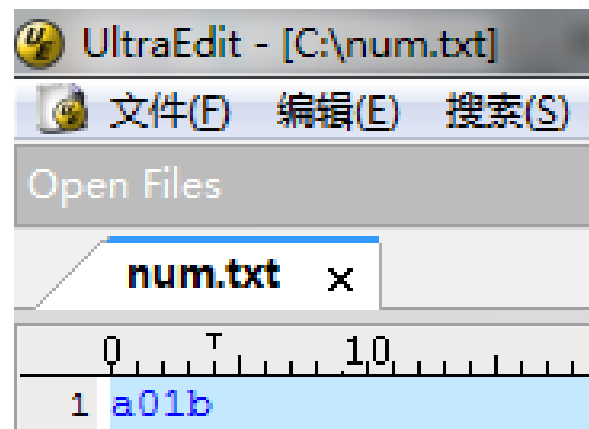
# fseek函数的补充说明

- **位移量**指以起始点为基点，向前移动的字节数。位移量应是long型数据(在数字的末尾加一个字母L)。
- fseek函数**一般用于二进制文件**。因为文本文件需要字符转换，fseek计算的位置往往会出错。
- 下面是fseek函数**调用**的几个例子：
  - ◆ `fseek (fp,100L,0);` //从文件头向后移100字节
  - ◆ `fseek (fp,50L,1);` //从当前位置向后移动50字节
  - ◆ `fseek (fp,-10L,2);` //从文件尾向前移动10个字节

# 把文件位置标记移到特定位置

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

3. int main()
4. {
5.     FILE * fp;
6.     fp = fopen("c:\\num.txt", "r+");
7.     if (fp==NULL)
8.         exit(0);
9.     fprintf(fp, "%c", 'a');
10.    fseek(fp, -1, SEEK_END);
11.    fputc('b', fp);
12.    fclose(fp); //关闭文件
13.
14.    return 0;
15. }
```



# 用ftell测定文件位置标记的位置

- 函数ftell()用来得到文件位置指针离文件开始处的偏移量，也就是可以获得文件的当前读/写位置。函数原型：

**long** ftell(FILE \*pfile);

- ◆ 该函数可以得到流文件当前的读/写位置，返回值是当前读/写位置偏离文件头部的字节数。
- ◆ 返回值是-1时表示执行出错。例如

```
i=ftell(fp); if(i== -1L) printf("error\n");
```

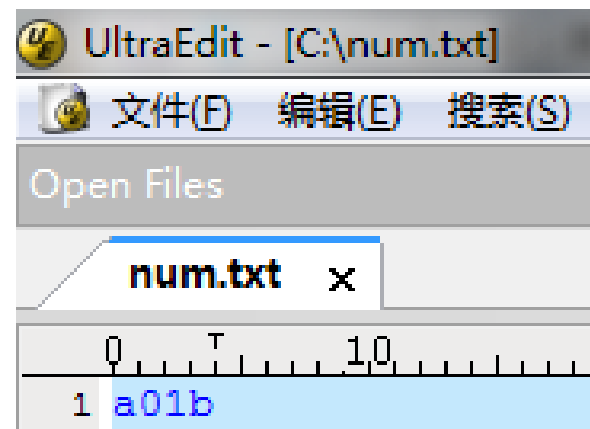


# 获得文件位置标记当前的位置

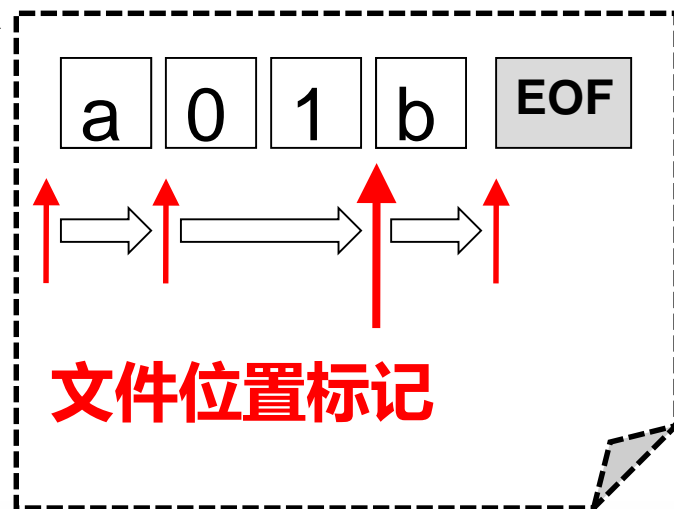
```
1. #include <stdio.h>
2. #include <stdlib.h>

3. int main()
4. {
5.     FILE * fp;
6.     int len;
7.     fp = fopen("c:\\num.txt", "r+");
8.     if (fp==NULL)
9.         exit(0);
10.    fprintf(fp, "%c", 'a');
11.    fseek(fp, -1, SEEK_END);
12.    len=ftell(fp);
13.    printf("%d\n", len);
14.    fputc('b', fp);
15.    fclose(fp); //关闭文件
16.    return 0;
17. }
```

3



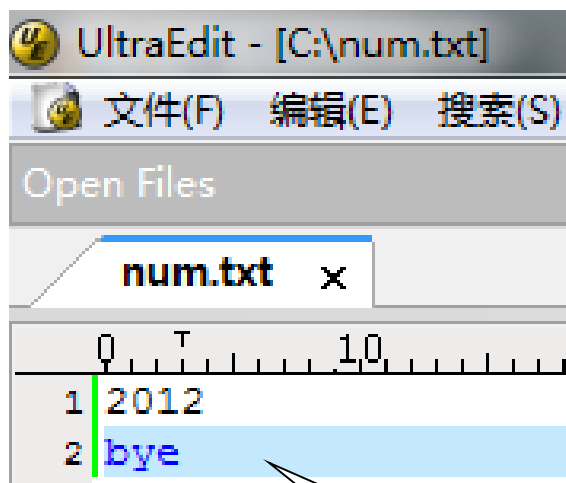
→  
fp



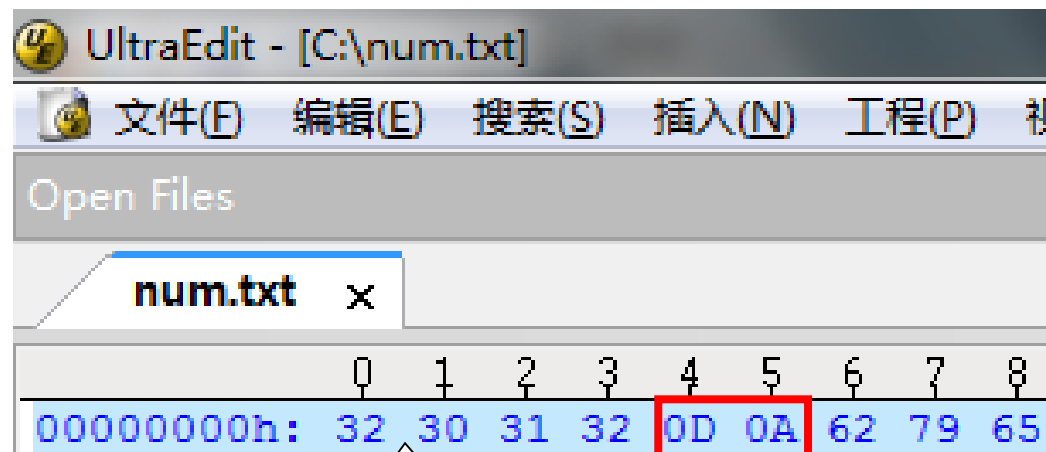
# 用fseek和ftell计算文件总长度

```
1. #include <stdio.h>
2. int main()
3. {
4.     FILE *pfile;
5.     long length;
6.     if ((pfile=fopen("file.dat", "rb"))==NULL)
7.         printf("Error on opening file!\n");
8.     else
9.     {
10.         fseek(pfile, 0L, SEEK_END);
11.         length = ftell(pfile);
12.         printf("the file has %ld bytes\n", length);
13.         fclose(pfile);
14.     }
15.     return 0;
16. }
```

# 文本文件中慎用fseek和ftell



文本模式



16进制模式

**因为：**有些操作系统中文本文件行末既有回车符又有换行符，另一些操作系统中则只有换行符

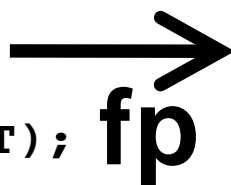
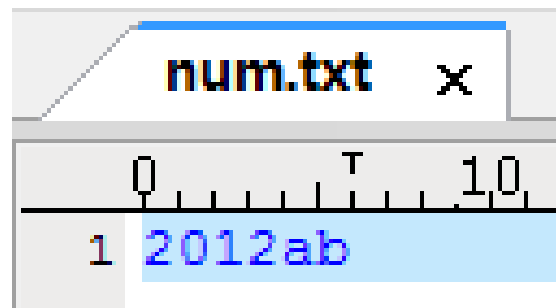
# "a+"模式的特别之处

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. int main()
4. {
5.     FILE * fp;
6.     char c;
7.     fp = fopen("c:\\num.txt", "a+");
8.     if (fp==NULL)
9.         exit(0);
10.    fputc('a', fp);
11.    fseek(fp, 0, SEEK_SET);
12.    c = fgetc(fp);
13.    printf("%c\n", c);
14.    rewind(fp);
15.    fputc('b', fp);
16.    fclose(fp); //关闭文件
17.    return 0;
18. }
```

读取结果

2

写入结果



fp

2	0	1	2	EOF
---	---	---	---	-----



打开文件时位置标记

# “a”与“a+”模式注意事项

- 当使用 “a” 或者 “a+” 模式打开文件时，所有的写入操作都在文件的结尾进行，可以使用 `fseek` 和 `rewind` 对文件指针进行重定位，但是写入操作会将文件指针重新移至文件结尾，因此，**已经存在的数据不会被覆盖。**
- 当使用 “a+” 模式打开文件时，文件既可以读也可以写，如果需要，可以使用 `fseek` 或者是 `rewind` 来改变当前位置，但**文件位置指针移动仅对读操作有效。**

# 思考题分析 1-1

## ➤ 下列代码的运行结果是什么？

```
#include <stdio.h>
int main()
{
    int a[5]={1,2,3,4,5};

    int *ptr = (int*)&a+1;
    printf("%d,%d", *(a+1), ptr[-1]);

    return 0;
}
```

2, 5

# 思考题分析 1-2

## ➤ 下列代码的运行结果是什么？

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

int main()
{
    char a[1000];
    int i;
    for (i=0; i<1000; i++)
    {
        a[i] = -1-i;
    }
    printf("%d", strlen(a));
    return 0;
}
```

255

# 思考题分析 2-1

➤ 下列代码的运行结果是什么？为什么？

```
#include <stdio.h>
int main()
{
    double a=3.0;
    float b=4.3;
    char c='a';
    a++;
    b++;
    c++;

    printf("%lf %lf %c\n", a, b, c);

    return 0;
}
```

4.000000 5.300000 b



## 思考题分析 2-2

➤ 下列代码的运行结果是什么？为什么？

```
#include <stdio.h>

int main()
{
    int a=0,b=3,c=5,d;

    d=++a || ++b&&c;

    printf("%d  %d  %d  %d \n",a,b,c,d);

    return 0;
}
```

1 3 5 1

## 思考题分析 2-3

➤ 下列代码的运行结果是什么？为什么？

```
#include <stdio.h>

int main()
{
    int a=0,b=3,c=5,d;

    d=a++ || ++b&&(c=6);

    printf("%d  %d  %d  %d \n",a,b,c,d);

    return 0;
}
```

1 4 6 1

## 思考题分析 2-4

➤ 下列代码的运行结果是什么？输入为 PI=3.142 ✓

```
#include <stdio.h>
int main()
{
    char b, c;
    int a, d;

    scanf("PI=%d%c%c%d", &a, &b, &c, &d);
    printf("a=%d,", a);
    printf("b=%c,", b);
    printf("c=%d,", c);
    printf("d=%d,", d);
    return 0;
}
```

```
PI=3.142
a=3, b=., c=49, d=42,
```

# 大家来找碴2

HD

街机模式

挑战模式

对战模式

游戏教程

排行榜



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

```
2. int main()
```

```
3. {
```

```
4.     x = 3;
```

```
5.     y = 6;
```

```
6.     printf("%d\n", x+y);
```

```
7. }
```



忘记定义变量

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

```
2. int main()
```

```
3. {
```

```
4.     int x, y;
```

```
5.     x = 3;
```

```
6.     y = 6;
```

```
7.     printf("%d\n", x+y);
```

```
8. }
```



```
1.int a=6;  
2.float b=4.5;  
3.double c;  
4 scanf("%ld", &c);  
5.printf("%f,%d\n",a,b);
```



输入输出的数据的  
类型与用户指定的  
输入输出格式声明  
不一致

```
1.int a=6;  
2.float b=4.5;  
3.double c;  
4 scanf("%lf", &c);  
5.printf("%d,%f\n",a,b);
```



```
1.char c = 500;  
2.short s = 50000;  
3.int i = 500000000000;
```

**✖ 赋值不可超过数据类型  
的取值范围**

```
1.short c = 500;  
2.int s = 50000;  
3.double i = 500000000000;
```

char类型:

-128 ~ 127

short类型:

-32768 ~ 32767

int类型:

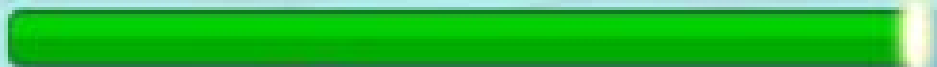
$-2^{31} \sim 2^{31}-1$



```
1.int a,b;  
2.char c[10];  
3.char str[3][20];  
4 scanf("%d%d", a, b);  
5 scanf("%s", c[0]);  
6 scanf("%s", str);
```

**✗ 使用scanf时忘记  
用变量的地址做参数**

```
1.int a,b;  
2.char c[10];  
3.char str[3][20];  
4 scanf("%d%d", &a, &b);  
5 scanf("%s", c);  
6 scanf("%s", str[0]);
```





```
1.int a,b;  
2 scanf("%d%d", a, b);
```

3, 4 ✓

 使用scanf时输入  
数据的格式与要求  
不符

```
1.int a,b;  
2 scanf("%d%d", a, b);
```

3 4 ✓

```
1.int a,b;  
2 scanf("%d,%d", a, b);
```

3,4 ✓



```
1.int a[10];  
2 scanf("%d", a);
```

```
1.int a[10];  
2.int i;  
3.for (i=0; i<10; i++)  
4.{  
5.    scanf("%d",&a[i]);  
6.}
```

**✗ 使用scanf向数值型  
数组输入数据时，  
用数值型数组名**



```
1. A = 3  
2. B = 4;
```

```
1. A = 3;  
2. B = 4;
```

**×** 语句后面漏掉分号

**注意**，此时编译器往往提醒错误语句的下一行出错了。



```
1.struct ppl
2.{
3.  char sex;
4.  int age;
5.}
```

```
1.struct ppl
2.{
3.  char sex;
4.  int age;
5.};
```



结构体定义后面  
漏掉分号



```
1. #include <stdio.h>;  
2. #define pi 3.14;
```

```
1. #include <stdio.h>  
2. #define pi 3.14
```

 预处理指令后面  
不应加分号



```
1.int a=10, b=5;  
2.if (a>b);  
3.    printf("a>b");
```

```
4.while (a>b);  
5.    b++;
```

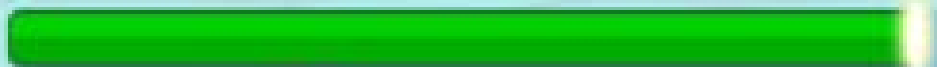
```
6.for (i=5; i<a; i++);  
7.    printf("%d\n", i);
```

**✗ if,while,for判断语句  
后面不应加分号**

```
1.int a=10, b=5;  
2.if (a>b)  
3.    printf("a>b");
```

```
4.while (a>b)  
5.    b++;
```

```
6.for (i=5; i<a; i++)  
7.    printf("%d\n", i);
```



```
1.int sum=0, i=1;
```

```
2.while (i<=100)
```

```
3.    sum = sum+i;
```

```
4.    i++;
```

```
1.int sum=0, i=1;
```

```
2.while (i<=100)
```

```
3.{
```

```
4.    sum = sum+i;
```

```
5.    i++;
```

```
6.}
```

**✗ 复合语句中多于一条  
语句时，应加花括号**

```
1.int i,count;  
2.int a[5]={1,2,3,4,5};  
3.for(i=0;i<5;i++)  
4.{  
5.    if (a[i]%2)  
6.        count++;  
7.}
```

```
1.int i,count=0;  
2.int a[5]={1,2,3,4,5};  
3.for(i=0;i<5;i++)  
4.{  
5.    if (a[i]%2)  
6.        count++;  
7.}
```

**❌ 累计，累加或累乘时  
变量忘记初始化。**





```
1. char c;
```

```
2. while((c=getchar())!='#')
```

```
3. {
```

```
4.     putchar(c);
```

```
5. }
```



括号不配对

```
1. char c;
```

```
2. while((c=getchar() )!='#')
```

```
3. {
```

```
4.     putchar(c);
```

```
5. }
```



```
1.int A, B, C;  
2.a = 2;  
3.b = 3;  
4.c = a + b;
```



使用标识符时，  
混淆了大小写字符

```
1.int a, b, c;  
2.a = 2;  
3.b = 3;  
4.c = a + b;
```

或者

```
1.int A, B, C;  
2.A = 2;  
3.B = 3;  
4.C = A + B;
```



```
1.int n=0, score=-1;
2.do
3.{
4.    if (score = 100)
5.    {
6.        n++;
7.    }
8.    scanf("%d",&score);
9.} while (score!=-1);
```

**✖ 误把 “=” 作为  
“等于” 运算符。**

```
1.int n=0, score=-1;
2.do
3.{
4.    if (score == 100)
5.    {
6.        n++;
7.    }
8.    scanf("%d",&score);
9.} while (score!=-1);
```



```
1. int i,a(10);
```

```
2. for (i=0; i<10; i++)
```

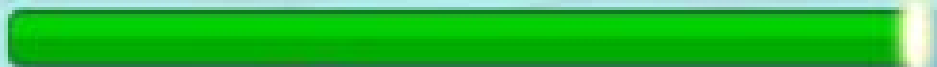
```
3.     scanf("%d", &a(i));
```

```
1. int i,a[10];
```

```
2. for (i=0; i<10; i++)
```

```
3.     scanf("%d", &a[i]);
```

**✗ 引用数组元素时，  
误用圆括号。**



```
1. int a[5]={1,2,3,4,5};  
2. int i;  
3. for (i=1;i<=5;i++)  
4.     printf("%d",a[i]);
```

```
1. int a[5]={1,2,3,4,5};  
2. int i;  
3. for (i=0;i<5;i++)  
4.     printf("%d",a[i]);
```



将数组元素个数  
误认为是可使用的  
最大下标值。



```
1. int a[5,4];
```

```
2. printf("%d",a[1+2,2+2]);
```

```
1. int a[5][4];
```

```
2. printf("%d",a[1+2][2+2]);
```



对二维或多维数组的  
定义和引用方法不对。



```
1. int a[3]={1,3,5}, b[3];
```

```
2. b = a;
```

```
1. int a[3]={1,3,5}, b[3];
```

```
2. int i;
```

```
3. for (i=0;i<3;i++)
```

```
4. {
```

```
5.     b[i]=a[i];
```

```
6. }
```



误以为数组名代表  
数组中全部元素。



```
1. char str[4];
```

```
2. str="Computer and C";
```

```
3. printf("%s\n", str);
```



混淆字符数组名  
与字符指针的区别。

```
1. char * str;
```

```
2. str="Computer and C";
```

```
3. printf("%s\n", str);
```





```
1. char * p;  
2. int * q;
```

```
3. scanf("%s", p);  
4. scanf("%d", q);
```



在引用指针变量前  
没有对它赋予确定  
的值。

```
1. char * p, c[20];  
2. int * q, i;
```

```
3. p = c;  
4. q = &i;
```

```
5. scanf("%s", p);  
6. scanf("%d", q);
```



```
1. switch(score)
2. {
3.     case 5:printf("super");
4.     case 4:printf("good");
5.     case 3:printf("pass");
6.     case 2:printf("fail");
7.     default: printf("error");
8. }
```

 **switch语句各分支  
中漏写break语句。**

```
1. switch(score)
2. {
3.     case 5:
4.         printf("super"); break;
5.     case 4:
6.         printf("good"); break;
7.     case 3:
8.         printf("pass"); break;
9.     case 2:
10.        printf("fail"); break;
11.     default:
12.        printf("error");
13. }
```



```
1. char sex;
```

```
2. sex="M";
```

```
1. char sex;
```

```
2. sex= 'M' ;
```



混淆字符和字符串的  
表示形式



```
1.int *p, a[3]={1,3,5};  
2.p = a;  
3.//输出a[1]  
4.printf("%d", *p++);
```

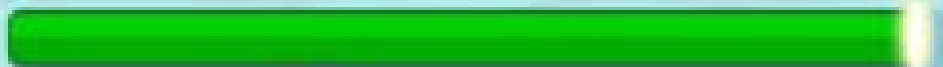


错误理解自加或自减  
运算符的执行顺序。

```
1.int *p, a[3]={1,3,5};  
2.p = a;  
3.//输出a[1]  
4.printf("%d", *++p));
```

或者

```
1.int *p, a[3]={1,3,5};  
2.p = a;  
3.//输出a[1]  
4.printf("%d", *(++p));
```



```
1. #include <stdio.h>
2. int main()
3. {
4.     float a=3.5,b=-7.6,c;
5.     c = max(a,b);
6.     printf("%f\n",c);
7.     return 0;
8. }
9. float max(float x, float y)
10. {
11.     return (x>y?x:y);
12. }
```

**❌ 忘记对调用发生在定义之前的函数进行声明。**

```
1. #include <stdio.h>
2. int main()
3. {
4.     float max(float,float);
5.     float a=3.5,b=-7.6,c;
6.     c = max(a,b);
7.     printf("%f\n",c);
8.     return 0;
9. }
10. float max(float x, float y)
11. {
12.     return (x>y?x:y);
13. }
```



```
1. #include <stdio.h>
2. int sum(int a[3])
3. {
4.     return a[0]+a[1]+a[2];
5. }
6. int main()
7. {
8.     int a[3]={1,3,5},b;
9.     b = sum(a[3]);
10.    printf("%f\n",b);
11.    return 0;
12.}
```

```
1. #include <stdio.h>
2. int sum(int a[3])
3. {
4.     return a[0]+a[1]+a[2];
5. }
6. int main()
7. {
8.     int a[3]={1,3,5},b;
9.     b = sum(a);
10.    printf("%f\n",b);
11.    return 0;
12.}
```

**✗ 当数组做函数形式参数时  
实参应传递指针或数组名。**



```
int fun(int x,float y,long z)
{
    ...
}
```

设有如上函数定义，以下声明都将出错：

```
fun(int x, float y, long z);
float fun(int x,float y,long z);
int fun(int x,int y,int z);
int fun(int x, float y);
int fun(int x,long z,float y);
```

 函数声明与函数定义  
不匹配。

```
int fun(int x,float y,long z)
{
    ...
}
```

设有如上函数定义，以下声明是**正确**的：

```
int fun(int x,float y,long z);
int fun(int, float, long);
int fun(int a,float b,long c);
```



```
int main()
{
    float f;
    scanf("%f", &f);
    printf("hello,");
    printf("%f\n", fabs(f));
    return 0;
}
```

```
#include <stdio.h>
#include <math.h>
int main()
{
    float f;
    scanf("%f", &f);
    printf("hello,");
    printf("%f\n", fabs(f));
    return 0;
}
```

 使用库函数时没有用  
#include包含头文件。





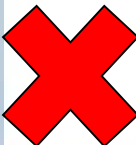
```
1. #include <stdio.h>
2. int main()
3. {
4.     int a=3,b=4;
5.     swap(a,b);
6.     printf("%d,%d\n",a,b);
7.     return 0;
8. }
9. void swap(int x,int y)
10.{
11.    int t;
12.    t=x; x=y; y=t;
13.}
```

 误以为函数形参值的  
改变会影响实参的值。

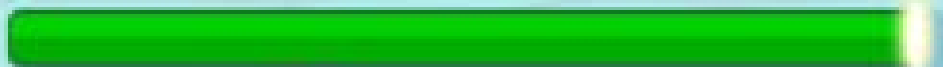
```
1. #include <stdio.h>
2. int main()
3. {
4.     int a=3,b=4,*p1,*p2;
5.     p1=&a; p2=&b;
6.     swap(p1,p2);
7.     printf("%d,%d\n",a,b);
8.     return 0;
9. }
10.void swap(int *p1,int *p2)
11.{
12.    int t;
13.    t=*p1; *p1=*p2; *p2=t;
14.}
```



```
1. #include <stdio.h>
2. int main()
3. {
4.     int fun(int x,int y);
5.     float a=3.5,b=4,6,c;
6.     c=fun(a,b);
7.     printf("%f\n",c);
8.     return 0;
9. }
10.int fun(int x,int y)
11.{
12.    return x+y;
13.}
```

 函数的实参和形参  
类型不一致。

```
1. #include <stdio.h>
2. int main()
3. {
4.     float fun(float,float);
5.     float a=3.5,b=4,6,c;
6.     c=fun(a,b);
7.     printf("%f\n",c);
8.     return 0;
9. }
10.float fun(float x,float y)
11.{
12.    return x+y;
13.}
```



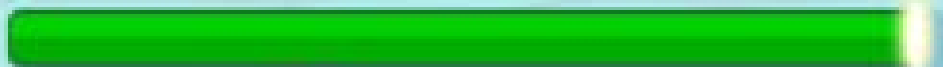
```
1. #include <stdio.h>
2. void main()
3. {
4.     printf("Hello\n");
5.     return 0;
6. }
```

 函数类型为void却在函数体内有返回值。

```
1. #include <stdio.h>
2. int main()
3. {
4.     printf("Hello\n");
5.     return 0;
6. }
```

或者

```
1. #include <stdio.h>
2. void main()
3. {
4.     printf("Hello\n");
5. }
```



```
1. int i=3, *p1;  
2. float a=1.5, *p2;  
3. p1 = &i;  
4. p2 = &a;  
5. p2 = p1;  
6. printf("%d,%d\n", *p1, *p2);
```

```
1. int i=3, *p1;  
2. float a=1.5, *p2;  
3. p1 = &i;  
4. p2 = &a;  
5. printf("%d,%f\n", *p1, *p2);
```

 不同类型的指针混用。



```
1. int i, a[5];  
2. for (i=0;i<5;i++)  
3. {  
4.     scanf("%d",a++);  
5. }
```

```
1. int i, a[5], *p;  
2. p = a;  
3. for (i=0;i<5;i++)  
4. {  
5.     scanf("%d",p++);  
6. }
```

或者

```
1. int a[5], *p;  
2. for (p=a; p<a+5; p++)  
3. {  
4.     scanf("%d",p);  
5. }
```

**✖ 混淆数组名与指针变量的区别。**



```
1. struct worker
2. {
3.     int num;
4.     char name[10];
5.     char sex;
6.     int age;
7. };
8. worker.num = 187045;
9. strcpy(worker.name, "tom");
10. worker.sex = 'M';
11. worker.age = 18;
```

**✗ 混淆结构体类型与结构体变量的区别。**

```
1. struct worker
2. {
3.     int num;
4.     char name[10];
5.     char sex;
6.     int age;
7. };
8. struct worker w1;
9. w1.num = 187045;
10. strcpy(w1.name, "tom");
11. w1.sex = 'M';
12. w1.age = 18;
```



```
1. struct pt
2. {
3.     int x;
4.     int y;
5. };
6. void draw(struct pt){ ... }
7. int main()
8. {
9.     struct pt p1;
10.    draw(struct pt);
11.    return 0;
12.}
```

**✗ 结构体做函数参数，  
调用应传递结构体变量**

```
1. struct pt
2. {
3.     int x;
4.     int y;
5. };
6. void draw(struct pt){ ... }
7. int main()
8. {
9.     struct pt p1;
10.    draw(p1);
11.    return 0;
12.}
```



# 思考题 3-1

➤ 下列代码的运行结果是什么？为什么？

```
#include <stdio.h>

int main()
{
    double *p = NULL;
    int a[100];
    printf("%d, %d\n", sizeof(p), sizeof(*p));
    printf("%d, %d\n", sizeof(a), sizeof(a[100]));
    printf("%d, %d\n", sizeof(&a), sizeof(&a[0]));

    return 0;
}
```



## 思考题 3-2

### ➤ 下列代码的运行结果是什么？

```
#include <stdio.h>
static int i,j; int k = 0;
int fun1()
{
    static int i=0; i++; return i;
}
void fun2()
{
    j=0; j++;
}
int main()
{
    for (k=0; k<10; k++) { i=fun1(); fun2(); }
    printf("%d,%d", i, j);
    return 0;
}
```

# 思考题 3-3

➤ 下列代码的运行结果是什么？

```
#include <stdio.h>

int main()
{
    int a[3][2] = {(0,1),(2,3),(4,5)};
    int *p;

    p = a[0];

    printf("%d", p[0]);

    return 0;
}
```

## 思考题 3-4

### ➤ 下列代码的运行结果是什么？

```
#include <stdio.h>

union myun
{
    struct { int x,y,z; }u;
    int k;
}a;

int main()
{
    a.u.x=4; a.u.y=5; a.u.z=6; a.k=0;
    printf("%d\n", a.u.x);

    return 0;
}
```

## 思考题 3-5

➤ 下列代码的运行结果是什么？

```
#include <stdio.h>

int main()
{
    union {int a; long b; unsigned char c; } m;

    m.b = 0x12345678;

    printf("%d\n", m.c);

    return 0;
}
```

# 思考题 3-6

➤ 下列代码的运行结果是什么？

```
#include <stdio.h>

union pw {int i; char ch[2]; } a;

int main()
{
    a.ch[0] = 2;
    a.ch[1] = 1;
    a.ch[2] = 0;
    a.ch[3] = 0;

    printf("%d\n", a.i);

    return 0;
}
```

# 思考题 3-7

## ➤ 下列代码的运行结果是什么？

```
#include <stdio.h>

typedef union
{
    long a[2];
    int b[4];
    char c[8];
} TY;

int main()
{
    TY our;
    printf("%d\n", sizeof(our));

    return 0;
}
```

# 思考题 3-8

➤ 下列代码的运行结果是什么？

```
void f(char *s)
{
    if (*s)
    {
        f(s+1);
        putchar(*s);
        f(s+1);
    }
}
int main()
{
    char *str="abc";
    f(str);
    return 0;
}
```

## 思考题 3-9

➤ 设数组a的首地址为200，以下程序输出是？

```
void main()
{
    short a[3][3]={1,2,3,4,5,6,7,8,9};
    printf("%d %d\n", a, *a);
    printf("%d %d\n", a[1], a+2);
    printf("%d %d\n", &a[0], &a[1][0]);
    printf("%d %d\n", a[1][0], *(a+1)+2);
}
```



# 思考题 3-10

- 假定当前路径下已有文件a1.txt，其内容如下：  
ABc#dEF 下列代码的运行结果是什么？

```
void f(FILE *fp)
{
    char c;
    while((c=fgetc(fp))!='#')
    {
        printf("%d ", c>='a'?c-'a'+10:c-'A'+10);
    }
}
int main()
{
    FILE *fp;
    fp=fopen("a1.txt","r");
    f(fp);
    fclose(fp);
    return 0;
}
```