

## 第二章

### 标准I/O

讲师：任继梅

# 课前提问

1. 什么是标准I/O终端？
2. 如何打开一个文件？
3. 如何进行字符读写？
4. 如何定位文件？
5. 如何关闭文件？

嵌入式

# 本章内容

## 2.1 文件的基本概念



### 2.1.1 文件的基本分类

### 2.1.2 File指针

### 2.1.3 标准I/O流

## 2.2 文件的常见操作



### 2.2.1 fopen/fclose

### 2.2.2 fgets/fputs

### 2.2.3 fgetc/fputc

### 2.2.4 fwrite/fread



# 本章目标

## ➤ 了解文件的基本概念



- 文件的基本分类
- File指针
- 标准I/O流

## ➤ 掌握文件的常见操作



- fopen/fclose
- fgets/fputs
- fgetc/fputc
- fwrite/fread

嵌入式

# 标准I/O知识点1-文件概念：

## ● 定义：

- 文件：一组相关数据的有序集合。
- 文件名：这个数据集合的名称。

## ● 按类型分类：

- 常规文件
  - ASCII码文件
  - 二进制的文件
- 目录
- 字符设备
- 块设备
- 有名管道
- 套接口
- 符号链接

嵌入式

# 文件概念

## ● 文件类型：

- 常规文件：这种文件包含了某种形式的数据
- 目录文件：包含其他文件的名字以及指向与这些文件有关的信息指针。
- 字符设备：这种类型提供对设备不带缓冲的访问，每次访问长度可变。
- 块设备：这种文件类型提供对设备带缓冲的访问，每次访问以固定长度为单位进行。
- FIFO：用于进程间通信
- 套接字：用于进程间的网络通信
- 符号链接：这种文件类型指向另一个文件。

# 标准I/O知识点2-流和file对象

- 不仅在UNIX系统，在很多操作系统上都实现了标准I/O库
- 标准I/O库由ANSI C标准说明
- 标准I/O库处理很多细节，如缓存分配、以优化长度执行I/O等，这样使用户不必关心如何选择合适的块长度
- 标准I/O在系统调用函数基础上构造的，它便于用户使用
- 标准I/O库及其头文件stdio.h为底层I/O系统调用提供了一个通用的接口。

# 标准I/O知识点2-流和file对象

## ● 文件指针

- FILE指针：每个被使用的文件都在内存中开辟一个区域，用来存放文件的有关信息，这些信息是保存在一个结构体类型的变量中，该结构体类型是由系统定义的，取名为FILE。
- 标准I/O库的所有操作都是围绕流(stream)来进行的，在标准I/O中，流用FILE \*来描述。
- 标准I/O库是由Dennis Ritchie在1975年左右编写的

## ● 流（stream）

- 定义：所有的I/O操作仅是简单的从程序移进或者移出，这种字节流，就称为流。
- 分类：文本流/二进制流。



# 标准I/O - 流和FILE对象

## ● 文本流

- 定义：在流中处理的数据是以字符出现。在文本流中，'\n'被转换成回车符 CR和换行符LF的ASCII码0DH和0AH。而当输出时，0DH和0AH被转成'\n'
- 数字2001在文本流中的表示方法为 '2' '0' '0' '1'  
ASCII: 50 48 48 49

## ● 二进制流

- 定义：流中处理的是二进制序列。若流中有字符，则用一个字节的二进制
- ASCII码表示；若是数字，则用对应的二进制数表示。对'\n'不进行变换
- 数字2001在二进制流中的表示方法为 00000111 11010001。

# 标准I/O 知识点3- 文件缓冲

## ● 缓冲文件系统(高级磁盘IO)

- 目的：尽量减少使用read/write的调用
- 定义：系统自动的在内存中为每一个正在使用的文件开辟一个缓冲区，从内存向磁盘输出数据必须先送到内存缓冲区，装满缓冲区在一起送到磁盘中去。从磁盘中读数据，则一次从磁盘文件将一批数据读入到内存缓冲区中，然后再从缓冲区逐个的将数据送到程序的数据区。
- 分类：全缓存，行缓存，不缓存。

## ● 非缓冲文件系统（低级磁盘IO）

- 定义：依靠于操作系统，通过操作系统的功能对文件进行读写，是系统级的输入输出。

# 缓存分类

## ● 标准I/O提供了三种类型的缓存

### ● 全缓存

- 当填满I/O缓存后才进行实际I/O操作，或者满足一定条件后，系统通过调用 malloc 来获得所需要的缓冲区域，默认值。
- 刷新(fflush)：标准I/O的写操作。
- 当缓冲区满了，或者满足一定的条件后，就会执行刷新操作。

### ● 行缓存

- 当在输入和输出中遇到换行符('\n')时，进行I/O操作。
- 当流遇到一个终端时，典型的行缓存。

### ● 不带缓存

- 标准I/O库不对字符进行缓冲，例如stderr。
- 很多的人机交互界面要求不可全缓存。
- 标准出错决不会是全缓存的。

## ● 使用setbuf()和setvbuf()可以更改缓存的类型

## ● 在任何时刻，可以使用fflush强制刷新一个数据流

## ● 定义：

```
#include <stdio.h>  
int fflush(FILE *fp);
```

## ● 说明：

- 可强制刷新一个流。此函数使该流所有未写的数据都被传递至内核。

# 标准I/O知识点4 - stdin,stdout,stderr

- 标准I/O预定义3个流，他们可以自动地为进程所使用

标准输入	0	STDIN_FILENO	stdin
标准输出	1	STDOUT_FILENO	stdout
标准错误输出	2	STDERR_FILENO	stderr

- 函数：int fprintf(FILE \* stream, const char \* format, ...);  
函数说明：fprintf()会根据参数format 字符串来转换并格式化数据，然后将结果输出到参数stream 指定的文件中，直到出现字符串结束('\0')为止。
- Stream 可以是表顺输入输出流
- 相关函数：printf, fscanf, vfprintf  
头文件：#include <stdio.h>

# stdin, stdout, stderr示例

```
#include <stdio.h>
main()
{
    int i = 150;
    int j = -100;
    double k = 3.14159;
    fprintf(stdout, "%d %f %x \n", j, k, i);
    fprintf(stdout, "%2d %*d\n", i, 2, i);
}
```

文件stdIOSample1.c，实现了从stdout到一个文本文件的重定向。即，把输出到屏幕的文本输出到一个文本文件中。

# stdin, stdout, stderr 示例

```
#include <stdio.h>
int main()
{
    int i;
    if (freopen("D:OUTPUT.txt", "w", stdout)==NULL)
        fprintf(stderr, "error redirecting\\stdout\\n");
    for(i=0;i<10;i++)
        printf("%3d",i);
        printf("\\n");
    fclose(stdout);
    return 0;
}
```

# 标准I/O知识点5-打开流

## ● 下列三个函数可用于打开一个标准I/O流：

- `FILE *fopen (const char *path, const char *mode);`
  - `FILE *freopen(const char *restrict pathname, const char *restrict type, FILE* restrict fp)`
  - `FILE *fdopen(int fildes, const char *type);`
- 
- `fopen()`打开由`path`指定的一个文件。
  - `mode`的值如下：



# 标准I/O - fopen() - mode参数

## ● 打开标准I/O流的mode参数：

r或rb	打开只读文件，该文件必须存在。
r+或r+b	打开可读写的文件，该文件必须存在。
w或wb	打开只写文件，若文件存在则文件长度清为0，即会擦些文件以前内容。若文件不存在则建立该文件。
w+或w+b或wb+	打开可读写文件，若文件存在则文件长度清为零，即会擦些文件以前内容。若文件不存在则建立该文件。
a或ab	以附加的方式打开只写文件。若文件不存在，则会建立该文件，如果文件存在，写入的数据会被加到文件尾，即文件原先的内容会被保留。
a+或a+b或ab+	以附加方式打开可读写的文件。若文件不存在，则会建立该文件，如果文件存在，写入的数据会被加到文件尾后，即文件原先的内容会被保留。

\* 当给定“b”参数时，表示以二进制方式打开文件。

# 标准I/O - fopen() - mode参数

## ● 打开一个标准I/O流的六种不同方式：

打开一个标准I/O流的六种不同的方式

限 制	r	w	a	r+	w+	a+
文件必须已存在	•			•		
擦除文件以前的内容		•			•	
流可以读	•			•	•	•
流可以写		•	•	•	•	•
流只可在尾端处写			•			•

# 标准I/O - fopen() - example

## ● stdIOSample2.c:

- 以读写方式打开文件file\_3，如果该文件不存在，则创建。如果该文件已经存在，则长度截短为0。

## ● stdIOSample3c:

- 以读写方式打开文件test.c，如果该文件不存在，则报错。
- 如果该文件已经存在，改变进程的标准输出，改标准输出到test.c文件中。

# 标准I/O - fopen() - example

```
#include<stdio.h>
#include<string.h>
#include<errno.h>
Int main()
{
    FILE *fp;
    if((fp=fopen("test_file",w+))==NULL)
    {
        printf(stderr,"fopen() failed %s\n",strerror(errno));
        return -1;
    }
    fclose(fp);
    return 0;
}
```

# 标准I/O - fopen() - example

```
#include<stdio.h>
#include<string.h>
#include<errno.h>
Int main()
{
    FILE *fp;
    if((fp=fopen("test.c","w+"))==NULL)
    {
        printf(stderr,"fopen() failed %s\n",strerror(errno));
        return -1;
    }
    freopen("test.c","w+",stdout);
    printf("test string\n");
    fclose(fp);
    return 0;
}
```

# 标准I/O库-fopen()文件permission

- fopen()没有设定创建文件权限的参数,
- POSIX.1要求具有如下权限(0666或者-rw-rw-rw):
  - S\_IRUSR|S\_IWUSR|S\_IRGRP|S\_IWGRP|S\_IROTH|S\_IWOTH
- 用户可以通过umask修改文件存取的权限, 其结果为(0666 & ~umask)
- 文件打开练习题
  - 设计c程序stdIOExercise1.c, 实现以只读的方式打开文件in.txt,以只写方式打开out.txt,如果出现错误, 则打印错误信息

# 标准I/O知识点6-fclose()

## ● fclose()用于关闭一个已经打开的流:

```
int fclose(FILE *stream);
```

- fclose()调用成功返回0，失败返回EOF，并设置errno
- 在该文件被关闭之前，刷新缓存中的数据。如果标准I / O库已经为该流自动分配了一个缓存，则释放此缓存。
- 当一个进程正常终止时(直接调用exit函数，或从main函数返回)，则所有带未写缓存数据的标准I / O流都被刷新，所有打开的标准I / O流都被关闭。
- 在调用fclose()关闭流后对流所进行的任何操作，包括再次调用fclose()，其结果都将是未知的。

练习：设计程序stdIOExercise2.c利用标准I/O函数来测试当前系统最大能打开的文件个数。

# 标准I/O知识点6-读写流

- 调用fopen()成功打开流之后，可在三种不同类型的非格式化I/O中进行选择，对其进行读、写操作：
  - 每次一个字符的I/O。使用fgetc()/fputc()一次读或写一个字符，如果流是带缓存的，则标准I/O函数处理所有缓存。
  - 每次一行的I/O。使用fgets()和fputs()一次读或写一行。每行都以一个换行符终止。当调用fgets()时，应说明能处理的最大行长。
  - 直接I/O。fread()和fwrite()函数支持这种类型的I/O。每次I/O操作读或写某种数量的对象，而每个对象具有指定的长度。这两个函数常用于从二进制文件中读或写一个结构。



# 读写流结束判定

## ● feof()

- 判断文件是否结束，可用于二进制文件。

```
int cTemp;
while (!feof(fp) && !ferror(stdin)) {
    cTemp = fgetc(fp);
}
```

## ● EOF/feof()

- EOF文件结束的返回标志，一般使用方法：

```
int cTemp;
cTemp = fgetc(fp);
while (cTemp != EOF) {
    cTemp = fgetc(infile);
}
```

# 读写流-字符I/O-输入

## ● 以下三个函数可用于一次读一个字符:

```
#include<stdio>
```

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

```
int fgetc(FILE*stream);
```

```
int getchar();
```

- 三个函数的返回：若成功则为下一个字符，若已处文件尾端或出错则为EOF
- 函数getchar()等同于getc(stdin)
- 注意，不管是出错还是到达文件尾端，这三个函数都返回同样的值。为了区分这两种不同的情况，必须调用ferror()或feof()。
- getc()的实现是一个宏，而fgetc()是一个函数。返回值为int类型。

# 读写流-字符I/O-输入

## 检查文件出错函数：

```
#include<stdio>
```

```
int feof(FILE *stream);
```

```
int ferror( FILE*stream);
```

```
Void clearerror(FILE *stream);
```

在大多数的FILE对象的实现中，保留两个标志：

- 出错标志。
- 文件结束标志。

# 读写流-字符I/O-输出

- 以下三个函数可用于一次输出一个字符:

<code>#include&lt;stdio&gt;</code>
<code>int putc(int c,FILE *stream);</code>
<code>int fputc(int c, FILE*stream);</code>
<code>int putchar(int c);</code>

- `putchar(c)`等价于`putc(c,stdout)`。出错返回EOF。
- `getc()/getchar()/putc()/putchar()`实现为宏，  
`fgetc()/fputc()`实现为函数，请根据实际情况选择。

# 读写流-字符I/O-example

设计c程序stdIOSample4.c,

example: 循环从标准输入(stdin)逐个字符读入数据, 写入一个文件, 再把该文件内容读出显示在屏幕上。

//向文件写

```
ch=getchar();
while(ch!='\n')
{
    fputc(ch,fp);
    ch=getchar();
}
```

//从文件中读

rewind(fp); //将文件内部  
位置指针从末尾移向开头;

```
ch=fgetc(fp);
while(ch!=EOF)
{
    putchar(ch);
    ch=fgetc(fp);
}
```

# 读写流-行I/O-输入

## ● 下列两个函数提供每次输入一行的功能：

```
#include<stdio>
```

```
char *gets(const char *s);
```

```
char *fgets(const char *s, int size, FILE* stream);
```

- 两个函数返回：若成功则为buf，若已处文件尾端或出错则为null
- 这两个函数都指定了缓存地址，读入的行将送入其中。gets()从标准输入读，而fgets()则从指定的流读。
- 对于fgets()，必须指定缓存的长度n。此函数一直读到下一个换行符为止，但是不超过n-1个字符，读入的字符被送入缓存。该缓存以null字符结尾。如若该行，包括最后一个换行符的字符数超过n-1，则只返回一个不完整的行，而且缓存总是以null字符结尾。对fgets()的下一次调用会继续读该行。
- gets()与fgets()的另一个区别是，gets()并不将换行符存入缓存中。

# 读写流-行I/O-输出

## ● 下列两个函数提供每次输出一行的功能：

```
#include<stdio>
```

```
int puts(const char *s);
```

```
int fputs(const char *s, FILE*stream);
```

- 两个函数返回：若成功则为非负值，若出错则为EOF
- 函数fputs()将一个以null符终止的字符串写到指定的流，终止符null不写出。注意，这并不一定是每次输出一行，因为它并不要求在null符之前一定是换行符。通常，在null符之前是一个换行符，但并不要求总是如此。
- puts()将一个以null符终止的字符串写到标准输出，终止符不写出。但是，puts()然后将一个换行符写到标准输出。
- puts()并不像它所对应的gets()那样不安全。但是我们还是应避免使用它，以免需要记住它在最后又加上了一个换行符。如果总是使用fgets()和fputs(),那么就会熟知在每行终止处我们必须自己加一个换行符。

# 读写流-I/O-example

设计c程序stdIOSample5.c, 循环读取output.txt的内容, 并输出终端, 然后再output.txt 文件尾追加字符串 “C language”

//读字符串关键代码

```
if((fp=fopen("output.txt","r"))==NULL)
{
    printf("Cannot open file strike any
key exit!");
    getchar();
    exit(1);
}
fgets(str,11,fp);
printf("%s/n",str);
```

//追加字符串关键代码

```
if((fp=fopen("output.txt","a+"))
==NULL)
{
    printf("Cannot open
file strike any key exit!");
    getchar();
    exit(1);
}
printf("input a string:/n");
scanf("%s",st);
fputs(st,fp);
rewind(fp);
```



# 读写流-二进制I/O

## ● 下列两个函数以执行二进制I/O(direct I/O)操作:

```
#include<stdio>
```

```
size_t fread(void *ptr, size_t size,size_t nmemb,file *stream);
```

```
size_t fwrite(constvoid *ptr, size_t size,size_t nmemb,file *stream);
```

### 参 数

- ptr:用于接收数据的内存地址
- size:要读写的字节数，单位是字节
- nmemb:要进行读写多少个size字节的数据项,每个元素是size字节.
- stream:输入流-文件指针

### 返回值

- 实际读取的元素个数。如果返回值为0，则可能文件结尾或发生错误。从ferror和feof获取错误信息或检测是否到达文件结尾。

# 读写流-二进制I/O-example

设计c程序stdIOSample6.c, 建立一个test.dat文件, 向里面写3个数据, 数据类型如下列结构体所示, 从文件读取这三个数据, 输出到显示屏

//结构体定义

```
struct test
{
    char name[20];
    int size;
} wr[nmemb];
```

/关键代码

```
fp = fopen("test.dat", "w");
fwrite(wr, sizeof(struct test),
nmemb, fp);
fclose(fp);
fp = fopen("test.dat", "r");
fread(re, sizeof(struct test),
nmemb, fp);
fclose(fp);
```

# 标准I/O – 效率

## ● fgets()/fputs()/getc/putc调用代码

```
int
main(void)
{
    int    c;

    while ((c = getc(stdin)) != EOF)
        if (putc(c, stdout) == EOF)
            err_sys("output error");

    if (ferror(stdin))
        err_sys("input error");

    exit(0);
}
```

```
int
main(void)
{
    char    buf[MAXLINE];

    while (fgets(buf, MAXLINE, stdin) != NULL)
        if (fputs(buf, stdout) == EOF)
            err_sys("output error");

    if (ferror(stdin))
        err_sys("input error");

    exit(0);
}
```

# 标准I/O – 效率

## ● fgets()/fputs()/getc/putc 效率对比图

**Figure 5.6. Timing results using standard I/O routines**

Function	User CPU (seconds)	System CPU (seconds)	Clock time (seconds)	Bytes of program text
best time from <a href="#">Figure 3.5</a>	0.01	0.18	6.67	
<code>fgets, fputs</code>	2.59	0.19	7.15	139
<code>getc, putc</code>	10.84	0.27	12.07	120
<code>fgetc, fputc</code>	10.44	0.27	11.42	120
single byte time from <a href="#">Figure 3.5</a>	124.89	161.65	288.64	

# 标准I/O库知识点7-定位流

## ● 定位标准I/O流的两种方式

- `ftell()`和`fseek()`: 这两个函数自V7以来就存在了，但是它们都假定文件的位置可以存放在一个长整型中。
- `fgetpos()`和`fsetpos()`。这两个函数是新由ANSI C引入的。它们引进了一个新的抽象数据类型`fpos_t`，它记录文件的位置。
- 需要移植到非UNIX系统上运行的应用程序应当使用`fgetpos()`和`fsetpos()`

# 定位流- fseek()/ftell()/rewind()

## ● fseek()/ftell()/rewind()函数原型:

<pre>#include&lt;stdio&gt;</pre>
<pre>Int fseek(FILE *stream, long offset,int whence);</pre>
<pre>long ftell(FILE *stream);</pre>
<pre>void rewind(FILE*stream);</pre>

# 定位流- fseek()/ftell()/rewind()

- ftell()用于取得当前的文件位置，调用成功则为当前文件位置指示，若出错则为-1L
- fseek()用户设定stream流的文件位置指示，调用成功返回0，失败返回-1，并设置errno
- fseek()的whence参数：SEEK\_SET/ SEEK\_CUR/ SEEK\_END。
- rewind()用于设定流的文件位置指示为文件开始，该函数调用成功无返回值。
- rewind()等价于(void)fseek(stream, 0L, SEEK\_SET)

# 定位流 - fgetpos()/fsetpos()

## ● fgetpos()/fsetpos()函数原型:

```
#include<stdio>
```

```
int fgetpos(FILE *stream, fpos_t *pos);
```

```
int fputpos(FILE *stream, fpos_t *pos);
```

- 两个函数返回：若成功则为0，若出错则为非0
- fgetpos()将文件位置指示器的当前值存入由pos指向的对象中。在以后调用fsetpos()时，可以使用此值将流重新定位至该位置。
- 需要移植的程序，应该优先考虑fgetpos()/fsetpos()。



# 标准I/O知识点8-临时文件

- 标准I / O库提供了两个函数以帮助创建临时文件：

```
#include<stdio>
```

```
Char *tmpnam(char *s);
```

```
FILE *tmpfile(void);
```

- tmpnam()产生一个与现在文件名不同的一个有效路径名字符串。每次调用它时，它都产生一个不同的路径名。
- tmpnam()的s如果为NULL，则返回值存放到一个静态的区中。如果s不为NULL，则认为其指向长度至少为L\_tmpnam个数的字符数组中。所产生的文件名存放该数组中，也作为函数返回值返回。
- tmpfile()创建一个临时二进制文件(类型wb+)，在关闭该文件或程序结束时将自动删除这种文件。

# 标准I/O库-临时文件example

- 设计c程序stdIOSample7.c，建立一个临时文件。

```
int main()
{
    char tmpname[L_tmpnam];
    char *filename;
    FILE *tmpfp;
    filename = tmpnam(tmpname);
    printf("Temporary file name is: %s\n", filename);
    tmpfp = tmpfile();
    if(tmpfp)
        printf("Opened a temporary file OK\n");
    else
        perror("tmpfile");
    exit(0);}
```

# 练习题

● 设计程序stdIOExercise3.c: 编程读写一个文件test.txt, 每隔1秒向文件中写入一行数据, 类似这样:

- 1, 2014-4-30 15:16:422, 2014-4-30 15:16:43
- 该程序应该无限循环, 直到按Ctrl-C中断程序。
- 再次启动程序写文件时可以追加到原文件之后, 并且序号能够接续上次的序号, 比如:

1, 2014-4-30 15:16:42  
2, 2014-4-30 15:16:43  
3, 2014-4-30 15:19:02  
4, 2014-4-30 15:19:03  
5, 2014-4-30 15:19:04



# 练习题

- INI文件是很常见的一种配置文件。比如：

```
;Configuration of http
[http]
domain=www.mysite.com
port=8080
cgihome=/cgi-bin
;Configuration of db
[database]server = mysql
user = myname
password = toopendatabase
```



- 方括号括起来的部分是section名，之后则是各个key = value键值对。
- 等号两边可以有若干个空白字符（空格或Tab），也可以没有空白字符。
- 注释是以分号;开始的行。
- 一个section结束时至少有一个空行，也可以有连续几个空行，空行是仅包括0个或若干个空白字符（空格或Tab）的行。
- INI文件的最后一行后面可能有换行符也可能没有。

# 练习题

- 现在要求编程stdIOExercise4.c, 把INI文件转化为XML文件。

```
<!-- Configuration of http -->
```

```
<http>
```

```
  <domain>www.mysite.com</domain>
```

```
  <port>8080</port>
```

```
  <cgihome>/cgi-bin</cgihome>
```

```
</http>
```

```
<!-- Configuration of db -->
```

```
<database>
```

```
  <server>mysql</server>
```

```
  <user>myname</user>
```

```
  <password>toopendatabase</password>
```

```
</database>
```



# 课程总结

## ● 本节课程内容

- 文件的基本概念
- 文本流和二进制流
- 文件的分类
- 文件的打开和关闭
- 字符读写文件
- 文件定位

## ● 下节课程

- 线性表的特点
- 单链表
- 双链表
- 循环链表

嵌入式

# 联系方式

扣扣: 59189174  
手机: 13788919225

上 散