计算机程序设计

Computer Programming

文件操作

主讲：吴锋

目录
CONTENTS

◎ 文件的概念

- 各类信息最终是以文件形式存储在永久存储设备中（如硬盘、光盘、U盘等）
- 文件是一组相关数据的有序集合
- 操作系统及应用程序往往通过对文件的输入输出完成某项功能
- 操作系统将与主机相连的输入输出设备也视为文件，如键盘是输入文件，显示器是输出文件

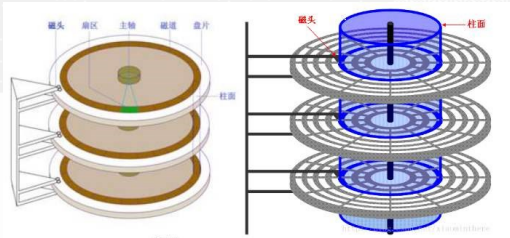
- 文件按结构形式分为文本文件和二进制文件
  - 文本文件是全部由字符组成（用ASCII码表示）的具有行列结构的文件，又称为ASCII码文件。便于对字符进行处理，可以直接显示
  - 二进制文件则是按数据在内存中的存储形式存储到文件中。一般不能直接显示
  - 文本文件的ASCII码也是二进制的。区别在于数值。例如，int型整数5在文本文件中保存的是0x35，而在二进制文件中是0x00,0x00,0x00,0x05四个字节

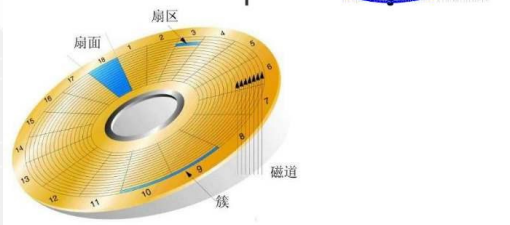
◎ 文件的概念

- 文件按数据的物理存取方式分为顺序文件和随机文件
  - 顺序文件：数据在存储介质中的实际顺序与它们进入存储器的顺序一致；一切存储在顺序存储器（如磁带）上的文件，都只能是顺序文件（难以在中间定位）
  - 随机文件：数据散列在存储介质中；硬盘上的文件，一般是随机文件（系统工具中的磁盘整理就是尽可能将分散随机存储的文件数据集中放置，以便于就近读写）

A记录	B记录	C记录

A记录		B记录	C记录	





◎ 文件的概念

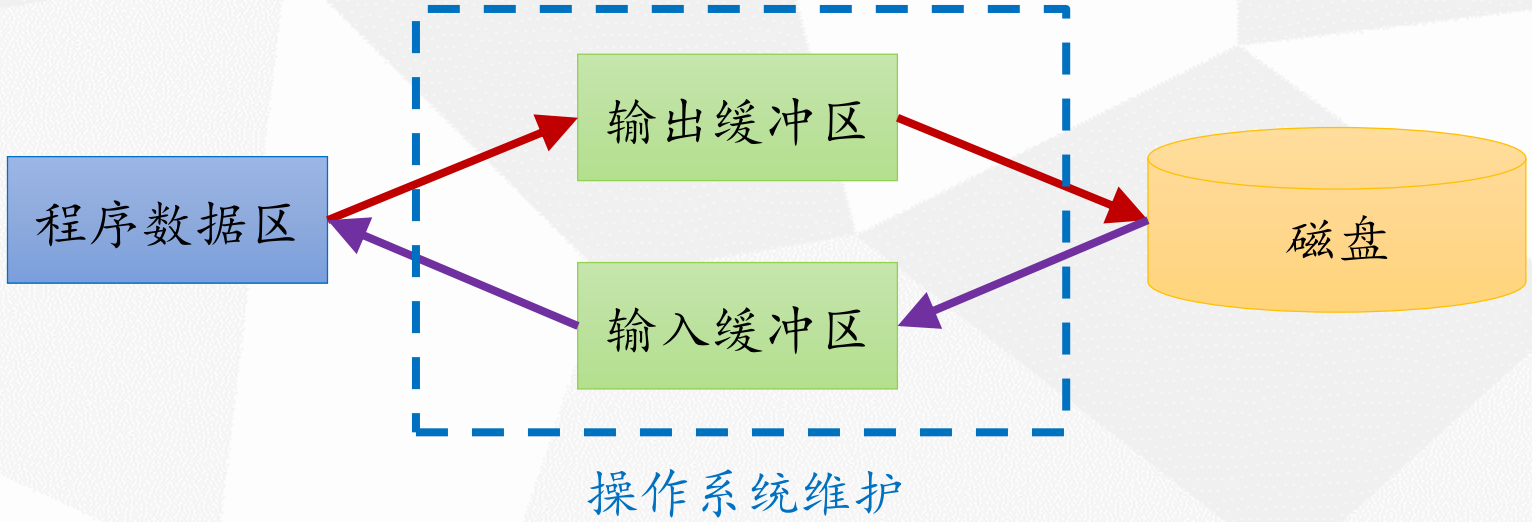
- 文件的读写方式分为顺序存取和随机存取
  - 顺序存取：从头到尾读写文件所有数据，文件被看成一个字符流，称为流式文件
    - 网络中所说的流是指一组有序数据序列，代表传输中的数字序列
    - 向文件顺序写入/读出数据
      - 打开文件并将文件指针置于文件的开头
      - 将内存中的数据顺序写入文件/将数据顺序读入内存
  - 随机存取：随机从文件内指定的位置读写数据
    - 向文件随机写入/读出数据
      - 打开文件并将文件定位指针置于待写入/读出的位置
      - 将内存中的数据写入文件/将数据读入内存
  - 顺序文件只能顺序存取，随机文件两种都可以

◎ 文件的概念

- 文件按存储的外部设备分为磁盘文件（普通文件）和设备文件
  - 磁盘文件是指存放在磁盘或其它外部介质上的有序数据集
  - 操作系统把与设备间的输入输出等同于对磁盘文件的读写，称其为设备文件，包括与主机相连的各种外部设备，如显示器、打印机、键盘等
  - 通常把显示器定义为标准输出文件(stdout)，键盘定义为标准输入文件(stdin)
    - 向stdout文件输出，即为在显示器上显示
    - 从stdin文件读入，即为从键盘读入

◎ 文件的概念

- 缓冲文件系统和非缓冲文件系统
  - 缓冲文件系统是指系统自动地在内存中为每一个正在使用的文件都开辟一个缓冲区
  - 从内存向磁盘输出数据时，缓冲文件系统先将数据送到内存中的缓冲区，待缓冲区满时才将整个缓冲区的数据一起保存到磁盘文件中（或强制写入）
  - 从磁盘向内存读入数据时，一般是从磁盘文件中一次读入一批数据到缓冲区，程序从缓冲区获取数据，缓冲区为空时再次读入一批数据



◎ 文件的概念

- 缓冲文件系统和非缓冲文件系统
  - 非缓冲文件系统是指系统不自动开辟缓冲区，而是由使用文件的程序直接操作文件数据，自行开辟和维护缓冲区
  - 现代操作系统和运行环境都采用缓冲文件系统
    - 写入时不要遗漏缓冲区内尚未存入的数据
    - 读取时可能不是“即时”输入的数据

◎ 文件类型与文件指针

- 操作系统读写文件时需要的信息
  - 文件当前的读写位置
  - 与文件对应的内存缓冲区地址
  - 文件的操作方式，等
- 以上信息都存放在“文件信息区”中，一个由系统定义的结构体类型变量，该结构体类型在stdio.h中定义，标识符是FILE，具体定义与编译系统有关

文件类型与文件指针

- FILE结构体示例

```
typedef  struct {
    int  _fd;          //文件号
    int  _cleft;       //缓冲区中剩余的字符
    int  _mode;        //文件操作模式
    char *_nextc;      //下一个字符位置
    char *_buff;       //文件缓冲区位置
} FILE;
```

- 缓冲文件系统中，关键的概念是“文件指针”，定义文件指针的一般形式：

```
FILE  *fp1,*fp2;
```

- 通过文件指针可以找到存放该文件信息的结构变量，然后按结构变量提供的信息找到文件并进行操作

◎ 文件的打开

- 文件需要打开才能操作，打开文件的实质是在用户程序和操作系统间建立起联系，通过文件指针共享文件信息
- ANSI C使用标准库函数fopen()打开或新建文件，函数原型：

  FILE *fopen(const char*fname, const char*mode);

  - fname是将要访问的文件名
  - mode是文件模式，用以规定文件可以操作的方式
  - FILE *是函数返回类型
    - 注意这里返回的是一个指针，文件的实质结构体变量由系统产生和维护

文件的打开

- 文件操作模式mode

文件操作模式	意义
r	以只读方式打开文本文件，该文件应该存在
w	以只写方式建立文本文件，若文件存在则清除文件内容；若文件不存在则建立该文件
a	以添加的方式打开文本文件。若文件不存在，则会建立该文件；如果文件存在，则从文件尾开始读写

- ”rb”、”wb”、”ab”与”r”、”w”、”a”相似，但对象是二进制文件
- ”r+”、”w+”、”a+” 以读写的操作模式打开文本文件。此时人为移动文件指针将切换读写模式
- ”rb+”、”wb+”、”ab+” 以读写的操作模式打开二进制文件

文件的打开

- fopen()函数的功能是按照指定的文件模式打开或创建指定的文件
  - 操作成功时返回返回与文件相对应的结构变量指针，失败时返回空指针NULL，例：

```
FILE  *fp;
fp=fopen(“prg_1.txt”,“r”);   //以只读方式打开当前目录下的文件prg_1.txt
```

  - 一般应检查文件打开操作是否成功：

```
FILE  *fp;
if((fp=fopen(“prg_1.txt”,“r”))==NULL) {
      printf(“can’t open this file! \n”);
      exit(0);
}
```

文件的打开

- 例

```
FILE *fp1, *fp2, *fp3;
char filename[]="file3.dat";
// 以文本只读方式打开file1
if (!(fp1=fopen("file1", "r"))) {
    printf("Cannot Open This File!\n");
    exit(0); // 退出程序
}
// 以二进制读写方式打开FILE2.TXT
fp2=fopen("C:\\HOME\\FILE2.TXT", "rb+");
// 以二进制读写方式打开file3.dat
fp3=fopen(filename, "a+b");
```

注意路径中的反斜线需要用字符转换

顺序并不严格

文件的关闭

- 打开文件后，系统会分配文件的缓冲区，文件使用完后应及时关闭文件以释放缓冲区
- ANSI C使用库函数fclose关闭文件，原型：

  int fclose(FILE *stream);

  - stream是打开文件时获得的文件指针（流指针）
  - 文件正常关闭后返回值为0，否则返回EOF即-1
  - 示例：fclose(fp);
- 关闭文件还有一个重要作用是操作系统此时将文件缓冲区的剩余内容写入文件，正常关闭文件才能确保文件内容的正确

◎ 实践技巧

- 在实践中可以使用宏，确保文件正常打开，且使用后关闭

```
#include <stdio.h>
#define OPEN_FILE(name, mode, fp) \
    for ( FILE *fp = fopen(name, mode), *_flag_ = (void*)0; \
        (fp == NULL ? (printf("Cannot open file: %s.\n", name), exit(0), 0) : !_flag_); \
        (fclose(fp), _flag_ = (void*)1) )

OPEN_FILE("file.txt", "r", fp) {
    ... ... // 对文件的操作
}
```

功能类似 Python 中的：

```
with open("file.txt", "r") as fp:
    ... ... # 对文件的操作
```

◎ 实践技巧

- 在实践中也可以使用宏，确保动态分配的内存被回收

```
#include <stdlib.h>
#define ALLOC_MEM(T, num, p) \
    for ( T *p = (T*)malloc(num*sizeof(T)), *_flag_ = (void*)0; \
        (p == NULL ? (printf("Cannot allocate memory.\n"), 0) : !_flag_); \
        (free(p), _flag_ = (void*)1) )

ALLOC_MEM(int, 1, p) {
    *p = 5; // 对指针的操作
}
```

◎ 实践技巧

- 在实践中也可以使用宏，实现方便的代码计时

```
#include <time.h>
#define TIMER \
    for ( clock_t _t_ = clock(), _flag_ = 0; !_flag_; \
        (printf("Time: %lf\n", (double)(clock()-_t_)/CLOCKS_PER_SEC), _flag_ = 1) )

TIMER {
    ... ... // 需要计时的代码段
}
```

◎ 文件的读写

- 常用的文件读写函数
  - 格式化读写函数 fscanf() 和 fprintf()，主要用于文本文件的读写
  - 字节（字符）读写函数fgetc()和fputc()，既适用于文本文件的读写，也适用于二进制文件的读写
  - 字符串读写函数fgets()和fputs()，用于文本文件读写
  - 数据块读写函数fread()和fwrite()，主要用于二进制文件的读写

- 注意：用文本或二进制方式打开文件后，并不能限制是用文本还是二进制形式读写数据，但结果会有差别（写值或写ASCII码，读换行符）

◎ 文件的读写

- 格式化（文本形式）读写函数

  int fscanf(FILE *stream, const char*format,...);

  int fprintf(FILE *stream, const char*format,...);

- 用法类似scanf与printf，但增加文件指针作为第一个形参
- 调用格式

  fscanf(文件指针, 格式字符串, 输入表列);

  fprintf(文件指针, 格式字符串, 输出表列);

```
fscanf(fp,“%d%f”,&i,&f);     //从指针fp所指的文件中读取一个整型
                             //数据到整型变量i中，再读取一个浮
                             //点型数据到浮点型变量f中

fprintf(fp,“%d”,i);          //将整型变量i的值存储到fp所指文件中
```

◎ 文件的读写

- 数据块（二进制形式）的读写函数

  size_t fread(void*buf, size_t size, size_t n, FILE*stream);

  size_t fwrite(const void*buf, size_t size, size_t n, FILE*stream);

  size_t是stdio.h中定义的类型(typedef unsigned int size_t;)

  - buf为输入/输出在内存中存放的首地址
  - size为读/写的字节数，即数据块的大小
  - n为输入/输出的数据项个数
  - stream为文件指针

文件的读写

- 数据块（二进制形式）的读写函数
  - 调用形式

    fread(buf, size, n, fp);

    从fp所指文件读入n个大小为size的数据块，存入buf所指内存，返回读取的数据项个数

    fwrite(buf, size, n, fp);

    从buf所指数据区取size*n个字节写入fp所指文件，返回写到文件中的数据项个数

  - 示例（用于结构变量的读写）

```
fread(&stu[i], sizeof(struct student), 1, fp);
fwrite(&stu[i], sizeof(struct student), 1, fp);
```

  - 示例（用于数组的读写）

```
fread(stu, sizeof(struct student), 3, fp);
fwrite(stu, sizeof(struct student), 3, fp);
```

文件的读写

- 字符读写函数

  int fgetc(FILE*stream);

  int fputc(int c,FILE*stream);

  - 每次读或写一个字节的数据，文本和二进制文件皆可，执行结果与文件打开方式有关（扩展字符集）
  - fread、fwrite函数以及字符串读写函数都通过此函数实现

- 例，

```
fp=fopen("test1.txt","r");
while ((ch=fgetc(fp))!=EOF) {....}
```

```
fp=fopen("test1.txt","rb");
while ((ch=fgetc(fp)),!feof(fp)) {....}
```

文件的读写

- 字符串读写函数

  char*fgets(char*str, int n,FILE*stream);

  - str指向用于存放读入字符串的存储空间地址，n表示最多能读取的字符个数
  - 从文件stream中读n-1个字符或遇到‘\n’或遇到EOF后加‘\0’放入缓冲区str
  - 调用成功返回str的首地址，失败返回NULL

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

  - s是字符串首地址，stream是文件指针
  - 向文件stream中写字符串s，但不将‘\0’写入文件
  - 写入成功返回字符个数，失败EOF

- 例，

```
fgets(str, n, fp);
fputs(“hello”, fp);
```

标准流指针

- C语言标准函数中，系统自动打开和关闭三个标准文件指针
  - 标准输入（键盘）stdin
  - 标准输出（显示器）stdout
  - 标准出错输出（显示器）stderr
- 下述输入/输出语句完全等价
  - scanf(char*fmt,...);　　fscanf(stdin,char*fmt,...);
  - printf(char*fmt,...);　　fprintf(stdout,char*fmt,...);
  - ch=getchar();　　ch=fgetc(stdin);
  - putchar(ch);　　fputc(stdout,ch);
  - gets(str,n);　　fgets(str,n,stdin);
  - puts(ch);　　fputs(ch,stdout);

文件读写位置指针

- 文件指针与文件读写位置指针
  - 文件指针
    - 重新赋值前保持不变，与待操作文件关联
    - 一般需要在程序中定义并赋值
  - 文件读写位置指针
    - 用以指示文件内部的当前读写位置，在打开或创建文件时由系统自动创建该指针
    - 每次对文件进行读写操作后该指针自动更新位置
    - 该指针是一个整数，表示当前位置（字节为单位）
- 文件读写位置指针的定位
  - 文件打开时，该指针位于文件开始或末尾（模式a）
  - 顺序读写文件时，该指针自动移动
  - 随机读写（不是从头顺序读写）时，需要先将读写位置指针移动到特定的位置，称为文件定位

文件定位

- 位置指针（重新）定位到开头

  void rewind(FILE *stream);

  - 应用例：打开已有文件，添加数据后从头读数据（模式a)

- 位置指针定位到指定位置（随机读写）

  int fseek(FILE *stream, long offset, int origin);

  - orgin为位置指针的移动参考点：
    - 文件头SEEK_SET（0）
    - 当前位置SEEK_CRU（1）
    - 文件尾SEEK_END（2）
  - offset为从参考点的位移量(字节数、长整型)

◎ 文件定位

- fseek函数的调用：
  - fseek(fp, 100L, SEEK_SET); //从文件头向后移动100个字节
  - fseek(fp, -10L, SEEK_CUR); //从当前位置向前移动10个字节
  - fseek(fp, -20L, SEEK_END); //从文件尾向前移动20个字节

- 如以读写模式打开文件，则每次调用fseek会 ~~改变位置指针的读写操作属性~~ 使用fflush(fp)函数将缓冲区中的内容写入文件中，并校正当前的文件指针。

◎ 文件定位

Section 7.21.5.3 of the C standard, which details the fopen function, states:

- When a file is opened with update mode (+ as the second or third character in the above list of mode argument values), both input and output may be performed on the associated stream.
- However, output shall not be directly followed by input without an intervening call to the fflush function or to a file positioning function (fseek, fsetpos, or rewind), and input shall not be directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file.
- Opening (or creating) a text file with update mode may instead open (or create) a binary stream in some implementations.

文件定位

- 例：

```
#include <stdio.h>
#include <stdlib.h>
void main() {
    FILE *fp; char ch;
    if ((fp=fopen("test4.txt","r+"))==NULL) {
        printf("can't open this file.\n");
        exit();
    }
    ch=fgetc(fp);
    while(!feof(fp)) {
        if (ch>='A' && ch<='Z') {
            ch+=32;
            fseek(fp,-1,SEEK_CUR);
            fputc(ch,fp);
            fseek(fp,0,SEEK_CUR);
        }
        ch=fgetc(fp);
    }
    fclose(fp);
}
```

读写方式打开，第一个操作是读

~~读写位置指针回移，切换为写操作~~

~~读写位置指针不动，切换为读操作~~ 将缓冲区内容写入文件

文件结束判断

- 文本文件：EOF(即-1)
  - 标准ASCII码表范围0-127
  - 128-255为扩展ASCII码表（有不同标准定义）
  - EOF(-1)的补码为0xFF，在扩展ASCII码表中为BLANK。读文本文件失败时返回-1

ASCII 字符代码表 一

高四位 / 低四位		ASCII非打印控制字符 0000 0					0001 1					ASCII 打印字符 0010 2		0011 3		0100 4		0101 5		0110 6		0111 7		
		十进制	字符	ctrl	代码	字符解释	十进制	字符	ctrl	代码	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	ctrl
0000	0	0	BLANK NULL	^@	NUL	空	16	►	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH	头标开始	17	◄	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX	正文结束	19	‼	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ	查询	21	§	^U	NAK	反确认	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK	确认	22	▬	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO	移出	30	▲	^6	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI	移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ	^Back space

注：表中的ASCII字符可以用:ALT + "小键盘上的数字键" 输入

ASCII 字符代码表 二

高四位 / 低四位		扩充ASCII码字符集 1000 8		1001 9		1010 A/10		1011 B/16		1100 C/32		1101 D/48		1110 E/64		1111 F/80	
		十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符
0000	0	128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
0001	1	129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
0010	2	130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
0011	3	131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
0100	4	132	ä	148	ö	164	ñ	180	┤	196	─	212	Ô	228	Σ	244	⌠
0101	5	133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
0110	6	134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	μ	246	÷
0111	7	135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
1000	8	136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
1001	9	137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	•
1010	A	138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
1011	B	139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
1100	C	140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	n
1101	D	141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	2
1110	E	142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
1111	F	143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	BLANK FF

注：表中的ASCII字符可以用:ALT + "小键盘上的数字键" 输入

文件定位

- 二进制文件及文本文件：使用feof()函数判断文件结束

  int feof(FILE*stream);

  - 文件读写位置指针移过了文件尾，返回非0值；否则返回0值
  - 用feof()判断文本文件结尾的时候，需要在取完最后一个字符后，再取一次，才知道越过了文件尾。所以即使是空文件，也需要fgetc()一次，然后feof()才能判断出到了文件尾

- 使用ftell()函数获取文件位置指针的当前位置

  long ftell(FILE*stream);

  - 调用成功则返回文件的位置指针，失败返回-1L

文件的错误检测

- 使用ferror()函数获取输入输出文件的错误

  int ferror(FILE*stream);

  - 若上一次读写函数调用出错，则返回非0值；否则返回0值
  - 调用读写函数后，可以根据读写函数返回值判断是否出错
  - 此外，系统会自动产生一个ferror()函数值
  - 可以在每次调用输入输出函数后，立即调用ferror()函数，获取错误信息
  - 执行fopen()后，ferror()的初始值自动置0

- 使用clearerr函数清除文件错误标志

  void clearerr(FILE*stream);

  - 清除文件错误标志ferror()和文件结束标志feof()
  - 只要读写发生错误，错误标志就会一直存在，直到对同一文件调用clearerr()函数、rewind()函数、或任何其他一个读写函数

◎ 作业

- 实现结构体那一章作业中的按书名检索的系统
  - 书籍数据从文件读入
  - 建立的数据库保存到文件中
  - 支持插入、删除、查找