西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

编程操作永久性数据

- 5、关闭文件
- 6、文件状态
- 7、文件缓存
- 8、文件定位

10.2.2 文件关闭

- fclose函数用来关闭已打开的文件，其函数原型为：

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

- 参数stream是已打开的文件指针。当文件关闭后，就不能再通过stream指针对原来关联的文件进行操作。除非再次打开，使该指针重新指向该文件。
- 应该使用fclose函数对不再操作的文件进行关闭。对缓冲文件系统来说，如果没有关闭文件而程序结束时，缓冲区的数据并没有实际写到文件中从而丢失。

10.2.2 文件关闭

- 标准输入输出库还提供了_fcloseall函数，可以将程序中所有打开的文件一次性关闭，其函数原型为：

```
int _fcloseall(void);
```

10.2.2 文件关闭

- 几乎所有文件应用中的打开与关闭的程序形式是相同的，为此给出通用的文件打开与关闭的步骤：
- ①定义文件指针变量。
- ②调用fopen打开文件（或创建文件）。
- ③打开文件失败时中断文件处理。
- ④继续进行文件各种操作。
- ⑤处理结束时关闭文件。

10.2.2 文件关闭

- 文件打开与关闭应用的代码通式：

```
FILE *fp; //定义文件指针变量
fp=fopen(文件名,操作模式); //打开文件或创建文件
if (fp!=NULL) { //打开或创建成功继续操作
   ......//文件各种操作
   fclose(fp); //处理结束时关闭文件
}
```

10.2.2 文件关闭

- freopen函数：用不同的文件或者模式重新打开文件。

```
FILE *freopen(const char*path, const char*mode, FILE *fp);
```

- path: 用于存储输入输出的文件名。
- mode: 文件打开的模式。和fopen中的模式相同。
- fp: 一个文件指针。

10.2.2 文件关闭

- freopen可用来实现重定向，所谓重定向是指把一个打开的文件重新定向到另外一个文件上。

- freopen可以将标准流（即stdin、stdout和stderr）重新定向到文件中。其中stdin是标准输入流，默认为键盘；stdout是标准输出流，默认为屏幕；stderr是标准错误流，一般把屏幕设为默认。

10.2.2 文件关闭

- freopen应用模式：

```
1  #include <stdio.h>
2  int main()
3  {
4    freopen("in.txt","r",stdin);//标准输入重定向到in.txt
5    freopen("out.txt","w",stdout);//标准输出重定向到out.txt
6    /* 以下标准输入输出操作均指向文件in.txt和out,txt
7      scanf printf
8      getchar putchar
9      gets puts
10   */
11   fclose(stdin);
12   fclose(stdout);
13   return 0;
14 }
```

10.2.3 文件状态

- 1．文件末尾检测
- 读取文件时需要检测是否已经到了文件末尾，以此判断文件读取工作是否应该结束。标准C语言提供了一个feof函数来判断是否已到文件末尾，其函数原型为：

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

- 参数stream是待检测的文件指针。如果文件已到末尾函数返回真（1），否则返回假（0）。

10.2.3 文件状态

- 2. 出错检测
- 在对文件进行各种操作时，可以用ferror函数检测操作是否出现错误，其函数原型为：

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

- 参数stream是待检测的文件指针。如果函数返回0，表示操作未出错，如果返回非零值表示出错。

10.2.3 文件状态

- 对文件每一次读写操作都会产生一个ferror函数值，通过立即检查ferror函数值就可以判断刚才调用的读写操作是否正确，并形成读写错误的程序处理策略。

10.2.3 文件状态

- clearerr函数可以是文件错误标志和文件结束标志置为0，其函数原型为：

```
void clearerr(FILE *stream);
```

- 其中参数stream是文件指针。

- 只要出现文件错误标志，就始终保留直到新的错误标志值出现。如果程序处理了这个错误，策略上来说就需要消除这个错误标志，以便正常的文件处理继续工作。

10.2.3 文件状态

- 3．错误类型
- 如果文件操作时出现错误，那么如何知道它是什么原因引起的呢？标准C语言为程序自动定义了一个全局整型变量errno来记录系统操作（包含文件输入输出）时发生的错误代码，其定义在头文件stdlib.h中，程序使用errno的形式为：

```
extern int errno;
//声明全局错误变量(已由标准C语言自动在别处定义)，引入到自己的程序中
```

- 根据errno值（操作系统设置）可以判断错误原因。

10.2.3 文件状态

【例10.1】

检测文件操作失败原因。

10.2.3 文件状态

例10.1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  extern int errno; //声明全局错误变量
4  int main()
5  {
6    FILE *fp; //定义文件指针变量
7    fp=fopen("test.dat","w"); //创建文件
8    if (fp!=NULL) { //创建成功继续操作
9      //文件各种操作
10     fclose(fp); //结束时关闭文件
11   }
12   else
13     switch (errno) { //根据错误值给出错误原因
14       case 13: printf("存取权限限制！"); break;
15       case 17: printf("文件已存在！"); break;
```

10.2.3 文件状态

例10.1

```
16          case 24: printf("打开太多文件！"); break;
17          case  2: printf("没有文件或目录！"); break;
18      }
19    return 0;
20 }
```

程序运行前在当前目录中新建文件test.dat，并且设置文件属性为只读，则程序运行结果为：

存取权限限制！

如果编写的是Windows程序，还可以调用GetLastError函数获取更详细的错误代码。

10.2.4 文件缓冲

- 文件操作时，有时需要清空缓冲区的数据，以便直接读入来自设备的数据。标准C语言提供了两个函数清空缓冲区，其函数原型为：

```
int fflush(FILE *stream); //清除指定文件stream的缓冲区
int _flushall(void); //清除所有文件的缓冲区
```

10.2.4 文件缓冲

- 例如在scanf("%d",&a); 后面紧接着执行getchar时，getchar不会等待字符输入，原因是scanf时输入的回车直接给了getchar。如果希望getchar等待输入，就需要在之前将键盘缓冲区清空，代码如下：

```
scanf("%d",&a); //键盘输入一个整型（以回车结束）
fflush(stdin); //清空标准输入文件（即键盘）缓冲区
ch=getchar(); //等待输入
```

10.4 文件定位

- 文件操作一般都是顺序读写，即每次读写完数据后，文件当前读写位置就自动移动到下一个数据位置。如果想改变这种顺序读写，使文件能够随机读写，就需要在文件读写前定位文件位置。

10.4 文件定位

- 1. 重置文件头
- rewind函数使文件位置重新回到文件开头，其函数原型为：

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

- 参数stream是已打开的文件指针。

- 2. 文件随机读写
- fseek函数使文件位置移到任意位置，其函数原型为：

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

- 参数stream是已打开的文件指针。参数origin表示移动时相对初始点，它必须是如下几个符号常量之一：

```
#define SEEK_SET   0 //移动时相对初始点为文件开头
#define SEEK_CUR   1 //移动时相对初始点为文件当前位置
#define SEEK_END   2 //移动时相对初始点为文件末尾
```

10.4 文件定位

- 参数offset表示移动的偏移量（相对于初始点），以字节为单位，如果为正值表示文件位置向前（远离文件头），为负值表示文件位置向后（趋向文件头）。例如：

```
fseek(fp,10,SEEK_SET);//文件位置设置为文件开头后第10个字节处
fseek(fp,10,SEEK_CUR);//文件位置设置为当前位置后第10个字节处
fseek(fp,-10,SEEK_CUR);//文件位置设置为当前位置前第10个字节处
fseek(fp,-10,SEEK_END);//文件位置设置为文件末尾前第10个字节处
fseek(fp,-10,SEEK_SET);//错误,origin为SEEK_SET时参数offset不能为负
fseek(fp,10,SEEK_END);//错误,origin为SEEK_END时参数offset不能为正
```

10.4 文件定位

【例10.7】

实现对一个文本文件内容的反向显示。

10.4 文件定位

例10.7

```
1  #include <stdio.h>
2  int main()
3  {
4    FILE *fp;
5    char ch;
6    fp=fopen("in.dat","r"); //打开文件读
7    if (fp!=NULL) {
8      fseek(fp,0L,2); //定位文件末尾即文件最后1个字符之后的位置
9      while ((fseek(fp,-1L,1))!=-1) { //相对当前位置退后1个字节
10       ch=fgetc(fp); //读取当前字符，文件指针会自动移到下一字符位置
11       putchar(ch); //显示
12       if (ch=='\n') //若读入是换行
13         fseek(fp,-2L,1); //换行为\r和\a，故要向前移动2个字节
14       else fseek (fp,-1L,1); //向前移动1个字节
15     }
```

10.4 文件定位

例10.7

```
16        fclose(fp); //关闭文件
17    }
18    return 0;
19 }
```

10.4 文件定位

- 3．文件当前位置
- ftell函数返回当前文件位置，其函数原型为：

```
long ftell(FILE *stream);
```

- 参数stream是已打开的文件指针。

10.4 文件定位

- 例如一个求文件大小函数getFileSize如下：

```
long getFileSize(const char *filename)
{
  long ret=-1;
  FILE *fp;
  fp=fopen(filename,"rb"); //打开文件读
  if (fp!=NULL) {
    fseek(fp,0,SEEK_END); //文件位置移到文件末尾
    ret=ftell(fp); //此时文件位置即是文件大小
    fclose(fp); //关闭文件
  }
  return ret; //返回负值表示文件不存在等错误
}
```


程序设计