第 3 章　文 件 操 作

程序执行时就称为进程，进程运行过程中的数据均在内存中。需要存储运算后的数据时，就需要使用文件。

学习目标：

- 理解文件存储机制。
- 掌握文件的打开、读写、关闭。

3.1　C 文件概述

文件是指存储在外部介质（如磁盘、磁带）上的数据集合。操作系统（Windows、Linux、Mac 等）是以文件为单位对数据进行管理的，如图 3.1.1 所示。

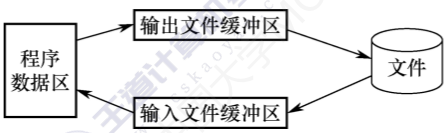

图 3.1.1　操作系统中的文件

C 语言对文件的处理方法如下。

缓冲文件系统：系统自动地在内存区为每个正在使用的文件开辟一个缓冲区。用缓冲文件系统进行的输入/输出称为高级磁盘输入/输出。

非缓冲文件系统：系统不自动开辟确定大小的缓冲区，而由程序为每个文件设定缓冲区。用非缓冲文件系统进行的输入/输出称为低级输入/输出。

下面介绍缓冲区原理。

缓冲区其实**就是一段内存空间**，分为读缓冲、写缓冲。C 语言缓冲的三种特性如下。

（1）**全缓冲**：在这种情况下，当填满标准 I/O 缓存后才进行实际 I/O 操作。全缓冲的典型代表是对磁盘文件的读写操作。

（2）**行缓冲**：在这种情况下，当在输入和输出中遇到换行符时，将执行真正的 I/O 操作。这时，我们输入的字符先存放到缓冲区中，等按下回车键换行时才进行实际的 I/O 操作。典型代表是标准输入缓冲区（stdin）和标准输出缓冲区（stdout）。

（3）**不带缓冲**：也就是不进行缓冲，标准出错情况（stderr）是典型代表，这使得出错信息可以直接尽快地显示出来。

3.2 文件的打开、读写、关闭

3.2.1 文件指针介绍

打开一个文件后，我们会得到一个FILE*类型的文件指针，然后通过该文件指针对文件进行操作。FILE是一个结构体类型，其具体内容如下所示：

```
struct _iobuf {
        char *_ptr;   //下一个要被读取的字符地址
        int    _cnt;   //剩余的字符，若是输入缓冲区，则表示缓冲区中还有多少个字符未被读取
        char *_base; //缓冲区基地址
        int    _flag; //读写状态标志位
        int    _file; //文件描述符
        int    _charbuf;
        int    _bufsiz; //缓冲区大小
        char *_tmpfname;
        };
typedef struct _iobuf FILE;
FILE *fp;
```

fp是一个指向FILE类型结构体的指针变量。可以使fp指向某个文件的结构体变量，从而通过该结构体变量中的文件信息来访问该文件。

Windows操作系统下的FILE结构体与Linux操作系统下的FILE结构体中的变量名是不一致的，但是其原理可以互相参考。

3.2.2 文件的打开与关闭

fopen函数用于打开由fname（文件名）指定的文件，并返回一个关联该文件的流。如果发生错误，那么fopen返回NULL。mode（方式）用于决定文件的用途（如输入、输出等），具体形式如下所示：

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

常用的mode参数及其各自的意义如下所示。

mode（方式）	意义
"r"	打开一个用于读取的文本文件
"w"	创建一个用于写入的文本文件
"a"	附加到一个文本文件
"rb"	打开一个用于读取的二进制文件
"wb"	创建一个用于写入的二进制文件
"ab"	附加到一个二进制文件

"r+"　　打开一个用于读/写的文本文件

"w+"　　创建一个用于读/写的文本文件

"a+"　　打开一个用于读/写的文本文件

"rb+"　　打开一个用于读/写的二进制文件

"wb+"　　创建一个用于读/写的二进制文件

"ab+"　　打开一个用于读/写的二进制文件

fclose 函数用于关闭给出的文件流，并释放已关联到流的所有缓冲区。fclose 执行成功时返回 0，否则返回 EOF。具体形式如下所示：

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

fputc 函数用于将字符 ch 的值输出到 fp 指向的文件中，如果输出成功，那么返回输出的字符；如果输出失败，那么返回 EOF。具体形式如下所示：

```
int fputc(int ch, FILE *stream);
```

fgetc 函数用于从指定的文件中读入一个字符，该文件必须是以读或读写方式打开的。如果读取一个字符成功，那么赋给 ch。如果遇到文件结束符，那么返回文件结束标志 EOF。具体形式如下所示：

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

下面通过例 3.2.1 来具体说明 fgetc 与 fputc 的使用。

【例 3.2.1】fgetc 与 fputc 的使用。

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

int main(int argc,char* argv[])
{
    FILE* fp;  //文件类型指针
    int i;
    char c;
    //printf("argc=%d\n",argc);
    //for(i=0;i<argc;i++)
    //{
    //    puts(argv[i]);
    //}
    fp=fopen(argv[1],"r+");
    if(NULL==fp)
    {
        perror("fopen");
```

```
        goto error;
    }
    //while((c=fgetc(fp))!=EOF)  //循环读取文件内容
    //{
    //    putchar(c);
    //}
    i=fputc('H',fp);
    if(EOF==i)
    {
        perror("fputc");
    }
    fclose(fp);
error:
    system("pause");
}
```

例 3.2.1 说明了对 main 函数传递参数的含义。假设编译后的可执行文件为 test.exe，执行 test.exe 时，后面跟的参数均是字符串，argv[i]依次指向每个元素，注意参数之间以空格隔开。例如 test.exe file1 file2，此时 argv[0]是 test.exe，argv[1]是 file1.txt，argv[2]是 file2.txt。注释的第一部分代码是打印 argv[0]、argv[1]、argv[2]等。那么如何设置项目进行传递参数呢？右键单击项目（注意是项目而不是解决方案），选择“属性”选项，得到如图 3.2.1 所示的界面。

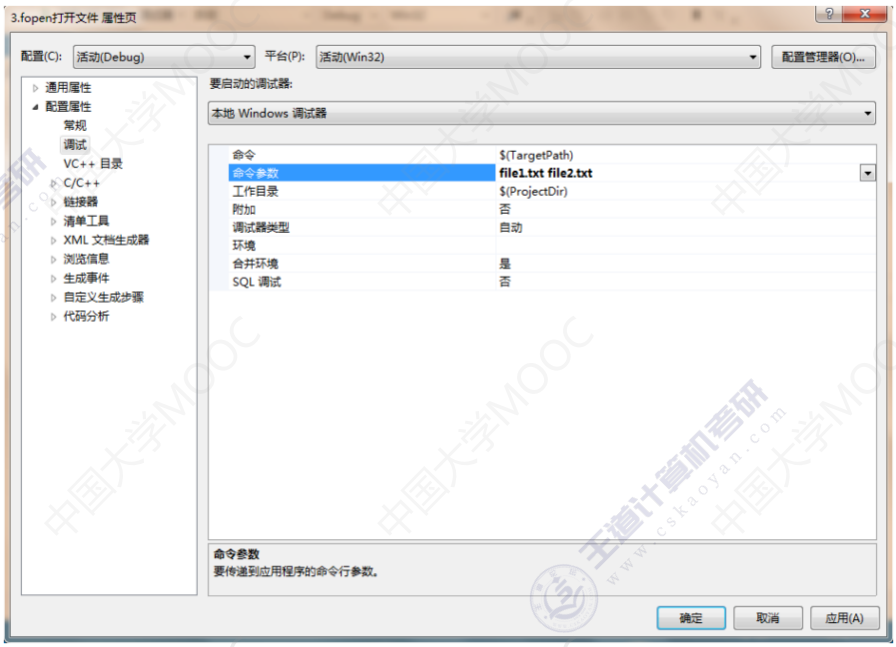

图 3.2.1 设置项目传递参数界面

在“调试”选项的“命令参数”一栏，输入 file1.txt、file2.txt 即可。

文件名用 argv[1]进行传递，打开文件后，得到文件指针 fp，如果文件指针 fp 为 NULL，那么表示打开失败，这时可用 perror 函数得到打开失败的原因（对于定位函数失败的原因，常用 perror 函数）。如果未新建一个文件，即文件不存在，那么会出现如图 3.2.2 所示的失败提示。

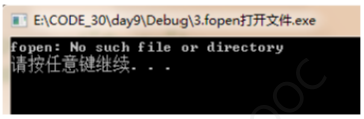

图 3.2.2　失败提示

冒号之前的内容是我们写入 perror 函数的字符串，冒号之后的内容是 perror 提示的函数失败原因，注意 perror 函数必须紧跟失败的函数，如果中间执行了 printf 这样的打印函数，那么 perror 函数将提示 Success，也就是没有错误，原因是每个库函数执行时都会修改错误码，一旦函数执行成功，错误码就会被改为零，而 perror 函数是读取错误码来分析失败原因的。

文件打开成功后，使用 fgetc 函数可以读取文件的每个字符，然后循环打印整个文件，读到文件结尾时返回 EOF，所以通过判断返回值是否等于 EOF 就可以确定是否读到文件结尾。上面这部分代码在例子中注释掉了，读者可以去掉注释，执行代码进行理解，注意要在自己新建的 file1.txt 文件中先填写一些内容。

为什么在执行 fputc 函数时注释掉了 fgetc 函数的内容呢？因为在 VS 中，读、写之间必须刷新光标（为便于理解，称为光标，实际上是位置指针），而刷新光标需要使用后面介绍的接口，所以用 fputc 函数将字符"H"写入文件时暂时注释掉了读取函数。

思考题：如果从文件中读到字符"m"时结束循环，那么应该如何修改？

3.2.3　fread 函数与 fwrite 函数

fread 函数与 fwrite 函数的具体形式如下：

```
int fread(void *buffer, size_t size, size_t num, FILE *stream);
int fwrite(const void *buffer, size_t size, size_t count, FILE *stream);
```

其中 buffer 是一个指针，对 fread 来说它是读入数据的存放地址，对 fwrite 来说它是输出数据的地址（均指起始地址）；size 是要读写的字节数；count 是要进行读写多少 size 字节的数据项；fp 是文件型指针；fread 函数的返回值是读取的内容数量，fwrite 写成功后的返回值是已写对象的数量。

fseek 函数的功能是改变文件的位置指针，其具体调用形式如下：

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

其中 fseek 的说明如下：

```
fseek(文件类型指针,位移量,起始点)
```

起始点的说明如下：

文件开头	SEEK_SET	0
文件当前位置	SEEK_CUR	1
文件末尾	SEEK_END	2

位移量是指以起始点为基点，向前移动的字节数。一般要求为 long 型。

fseek 函数调用成功时返回零，调用失败时返回非零。

下面来看例 3.2.2。

【例 3.2.2】fread 与 fwrite 及 fseek 的使用。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc,char** argv)
{
    char buf[20]="hello\nworld";
    FILE* fp;
    int i=12345;
    int ret;
    if(argc!=2)
    {
        printf("error args\n");
        return -1;
        system("pause");
    }
    fp=fopen(argv[1],"r+");
    if(NULL==fp)
    {
        perror("fopen");
        return -1;
    }
    //向文件中写入整型数
    //ret=fwrite(&i,sizeof(int),1,fp);
    //i=0;
    //ret=fread(&i,sizeof(int),1,fp);
    ret=fwrite(buf,sizeof(char),strlen(buf),fp); //把 buf 中的字符串写入文件
    memset(buf,0,sizeof(buf)); //清空 buf
    ret=fseek(fp,-12,SEEK_CUR); //往前偏移 12 字节
    ret=fread(buf,sizeof(char),sizeof(buf)-1,fp);
```

```
    puts(buf);//打印 buf 的内容
    fclose(fp);
    system("pause");
}
```

fread 和 fwrite 函数既可以以文本方式对文件进行读写，又可以以二进制方式对文件进行读写。以"r+"即文本方式打开文件进行读写时，向文件内写入的是字符串，写入完毕后会将 buf 清空，这时文件位置指针指向 12 字节的位置，如果要从文件头读取，那么就必须通过 fseek 函数偏移到文件头。例子中以当前位置为基准，向前偏移 12 字节，接着通过 fread 函数读取文件，读取内容后进行打印。为什么写入的是"hello\nworld"共 11 字节，而想偏移到文件开头却需要偏移 12 字节呢？这是因为在文本方式下，向文本文件中写入"\n"时实际存入磁盘的是"\r\n"，所有的接口调用都是 Windows 的系统调用，这是 Windows 的底层实现所决定的。当然，以文本方式写入，以文本方式读出，遇到"\r\n"时底层接口会自动转换为"\n"，因此用 fread 函数再次读取数据时，得到的依然是"hello\nworld"，共 11 字节。

如果把 fopen 函数中的"r+"改为"rb+"，也就是改为二进制方式，那么当我们向磁盘写入 11 字节时，磁盘实际存储的就是 11 字节，这时向前偏移时，fseek(fp,-12,SEEK_CUR);需要修改为 fseek(fp,-11,SEEK_CUR);，因为实际磁盘存储只有 11 字节。在二进制方式下，如图 3.2.3 所示，文件大小是 11 字节，如果这时双击打开该文件，那么会发现没有换行，即 helloworld 是连在一起的，中间没有换行符，原因是文本编辑器必须遇到"\r\n"时才进行换行操作。

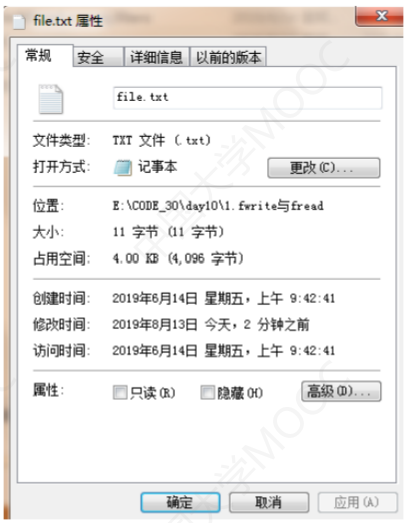

图 3.2.3　文件详细信息

相信读者此时已经理解了文本方式和二进制方式的差异。以文本方式下写入"\n"后，磁盘存储的是"\r\n"，当然读取时会以"\n"的形式读出"\r\n"。而以二进制方式写入"\n"后，磁盘存储的是"\n"。二者在其他方面没有差异。那么如何避免出错呢？如果是以文本方式写入的内容，那

么一定要以文本方式读取；如果是以二进制方式写入的内容，那么一定要以二进制方式读取，不能混用!

注释部分是二进制方式的演示，处于二进制方式时，需要以"rb+"方式打开文件，二进制方式下内存中存储的是什么，写入文件的就是什么，要保持一致。例如，写入整型变量 i，其值为 12345，内存存储为 4 字节，即 0x000004D2，那么写入内存的也是 4 字节。这时双击打开文件看到的是乱码，所以读取时也要用一个整型变量来存储。在实际工作中，我们往往以二进制方式来存储数据。

思考题：如果将代码中的 fseek(fp,-12,SEEK_CUR);改为 fseek(fp,-11,SEEK_CUR);，那么读取后，puts(buf)执行打印的效果是什么？如果用 fseek 偏移位置指针到文件头、文件尾呢？

3.2.4 fgets 函数与 fputs 函数

函数 fgets 从给出的文件流中读取[num-1]个字符，并且把它们转储到 str（字符串）中。fgets 在到达行末时停止，fgets 成功时返回 str（字符串），失败时返回 NULL，读到文件结尾时返回 NULL。其具体形式如下：

```
char *fgets(char *str, int num, FILE *stream);
```

fputs 函数把 str（字符串）指向的字符写到给出的输出流。成功时返回非负值，失败时返回 EOF。其具体形式如下：

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

下面来看例 3.2.3。

【例 3.2.3】fgets 与 fputs 的使用。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc,char** argv)
{
    char buf[20]={0};
    FILE* fp;
    int i=1234;
    int ret;
    char *p;
    if(argc!=2)
    {
        printf("error args\n");
        return -1;
        system("pause");
```

```
    }
    fp=fopen(argv[1],"r+");
    if(NULL==fp)
    {
        perror("fopen");
        return -1;
    }
    while(fgets(buf,sizeof(buf),fp)!=NULL)  //读取到文件结束时，fgets 返回 NULL
    {
        printf("%s",buf);
    }
    system("pause");
    return 0;
}
```

使用 fgets 函数，我们可以一次读取文件的一行，这样就可以轻松地统计文件的行数。同时，读取一行字符串后，我们可以按照自己的方式进行单词分割等操作。注意，在做一些在线评测题目时，用于 fgets 函数的 buf 不能过小，否则可能无法读取"\n"，导致行数统计出错。fputs 函数向文件中写一个字符串，不会额外写入一个"\n"。这个函数比较简单，读者可以自己编写代码看一下该函数的具体执行效果。

思考题：上面的代码执行 fgets(buf,sizeof(buf),fp)后未对 buf 进行清空，是否有影响？

3.2.5　ftell 函数

ftell 函数返回 stream（流）当前的文件位置，发生错误时返回–1。当我们想知道位置指针距离文件开头的位置时，就需要用到 ftell 函数，其具体形式如下所示：

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

下面来看例 3.2.4。

【例 3.2.4】ftell 与 fseek 的使用。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    FILE *fp;
    char str[20]="hello\nworld";
    int val=0;
    long pos;
```

```
    int ret;
    fp =fopen("file.txt","r+");
    if(NULL==fp)
    {
        perror("fopen");
        goto error;
    }
    val=strlen(str);
    ret=fwrite(str,sizeof(char),val,fp);
    ret=fseek(fp,-5,SEEK_CUR);
    if(ret!=0)
    {
        perror("fseek");
        goto error;
    }
    pos=ftell(fp);  //获取位置指针距离文件开头的位置
    printf("Now pos=%ld\n",pos);
    memset(str,0,sizeof(str));
    ret=fread(str,sizeof(char),sizeof(str),fp);
    printf("%s\n",str);
    fclose(fp);
error:
    system("pause");
}
```

最终的运行效果如图 3.2.4 所示。

我们向文件中写入了"hello\nworld"，因为是文本方式，所以总计为 12 字节，通过 fseek 函数向前偏移 5 字节后，用 ftell 函数得到的位置指针距离文件开头的位置即为 7，这时再用 fread 函数读取文件内容，得到的是"worl"。

图 3.2.4　例 3.2.4 中代码的运行效果

思考题：如果代码改为 fseek(fp,-11,SEEK_CUR)，那么 pos 打印得到的值为多少？