第 9 章 文件操作

（视频讲解：2.5 小时）

程序执行起来后，称之为进程，进程运行过程中的数据，均在内存中，当我们需要把运算后的数据存储下来时，就需要文件。学习本章，你将掌握：

- **理解文件分类及存储机制**
- **掌握文件打开，读写，关闭**

9.1 C 文件概述

文件：文件指存储在外部介质(如磁盘磁带)上数据的集合。操作系统（windows，Linux，Mac 均是）是以文件为单位对数据进行管理的（原理如图 9.1-1）。

图 9.1-1

文件的分类

●从用户观点:

特殊文件(标准输入输出文件或标准设备文件)。

普通文件(磁盘文件)。

●从操作系统的角度看，每一个与主机相连的输入输出设备看作是一个文件。

例：输入文件：终端键盘

输出文件：显示屏和打印机

文件的分类

●按数据的组织形式:

ASCII 文件(文本文件):每一个字节放一个 ASCII 代码

二进制文件:把内存中的数据按其在内存中的存储形式原样输出到磁盘上存放。

例：整数 10000 在内存中的存储形式以及分别按 ASCII 码形式和二进制形式输出如图 9.1-2 所示：

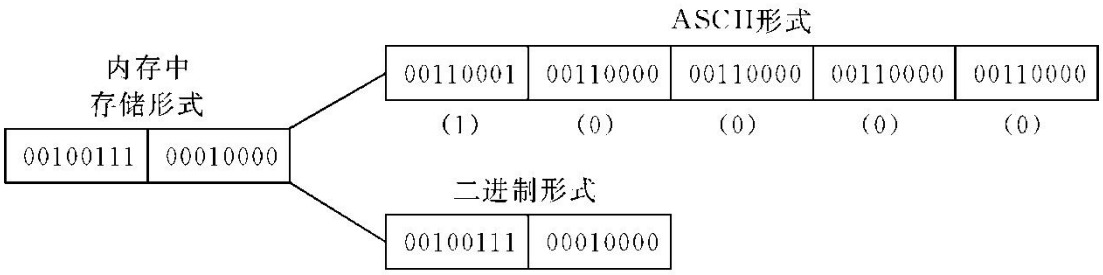


图 9.1-2

ASCII 文件和二进制文件的比较：

ASCII 文件便于对字符进行逐个处理，也便于输出字符。但一般占存储空间较多，而且要花费转换时间。

二进制文件可以节省外存空间和转换时间，但一个字节并不对应一个字符，不能直接输出字符形式。

这里很多同学不是非常理解 ASCII 文件和二进制文件的差异，不用担心，后面在 fread 与 fwrite 我们通过实例来理解差异会变的非常简单。

一般中间结果数据需要暂时保存在外存上，以后又需要输入内存的，常用二进制文件保存。后台开发数据往往以二进制形式保存。

C 语言对文件的处理方法：

缓冲文件系统：系统自动地在内存区为每一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又称为高级磁盘输入输出。（标准 C 是缓冲文件系统）

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又称为低级输入输出系统。

9.2 文件的打开、读写、关闭

9.2.1 文件指针介绍

打开文件后我们得到 FILE*类型的文件指针，通过该文件指针对文件进行操作，FILE 是一个结构体类型，那么首先让我们来看下它里边都有什么呢

```
struct _iobuf {
        char *_ptr;//下一个要被读取的字符地址
        int   _cnt; // 剩余的字符，如果是输入缓冲区，那么就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 指向某一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能够访问该文件。

只是 Windows 下的 FILE 结构体，Linux 下与 Windows 结构体里的变量名是不一致的，但是原理可以参考。

9.2.2 文件的打开与关闭

```
#include <stdio.h>
    FILE *fopen( const char *fname, const char *mode );
```

fopen()函数打开由 *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, fopen()返回 NULL. *mode*(方式)是用于决定文件的用途(例如 用于输入,输出,等等)

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

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

函数 fclose()关闭给出的文件流, 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.

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

将字符（ch 的值）输出到 fp 所指向的文件中去，如果输出成功，则返回值就是输出的字符；如果输出失败，则返回一个 EOF。

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

从指定的文件读入一个字符,该文件必须是以读或读写方式打开的,读取成功一个字符,赋给 c h 。如果遇到文件结束符，返回一个文件结束标志 EOF。

下面来看代码实例【例 9.2.2-1】：

【例 9.2.2-1】fgetc 与 fputc 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");
    }
```

实例【例 9.2.2-1】第一展示对于 main 函数传参的含义，我们编译出的可执行文件假如为 test.exe,那么执行 test.exe 后面跟的参数均为字符串，argv[i]依次指向每一个元素，注意每个参数之间以空格隔开，例如 test.exe file1 file2 那么 argv[0]是 test.exe, argv[1]是 file1.txt，argv[2]是 file2.txt，注释的第一部分代码，就是打印 argv[0],argv[1],argv[2]等每个参数的作用,如何设置项目进行传参呢？右键点击项目（注意是项目，不是解决方案），选择属性，得到如图 9.2.2-1 所示的效果：

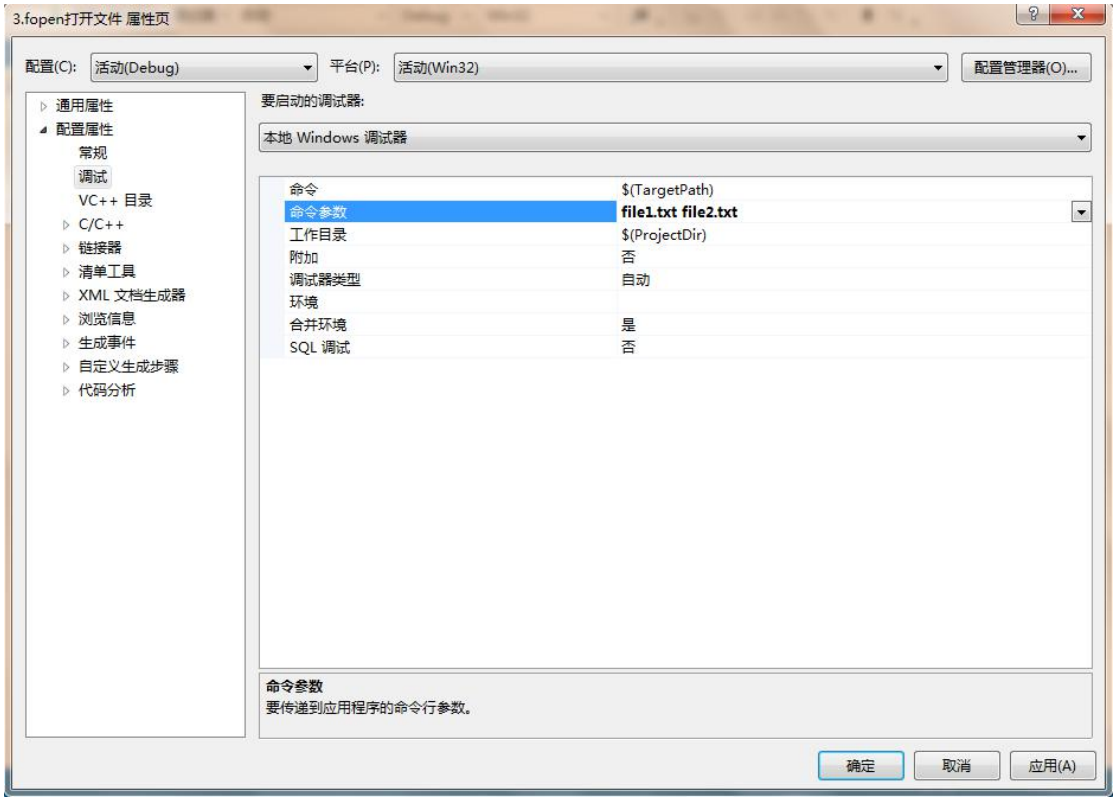
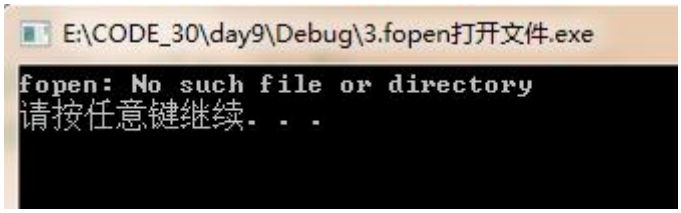


图 9.2.2-1

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，代表打开失败，我们通过 perror 函数可以得到打开失败的原因（对于定位函数失败原因，perror 非常常用）**。假如我们没有新建一个文件，即文件不存在，会得到如下失败提示：



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

文件打开成功后，通过 fgetc 可以读取文件的每一个字符，然后循环打印整个文件，读到文件结尾返回 EOF，所以通过判断是否等于 EOF，就可以确定是否读到文件结尾。上面这部分代码注释掉了，各位同学可以去除注释，编写代码进行理解，各位同学在自己新建的 file1.txt 中注意填写一些内容。

为什么在执行 fputc 时，注释了 fgetc 的内容呢，因为在 VS 中，读写之间必须刷新光标（为了大家理解称为光标，实际是位置指针），而刷新光标需要通过下面 9.2.3 讲的接口，所以通过 fputc 将字符大 H 写入文件时，我们注释了读取。

思考题：如果从文件中读到字符 m 时，让循环结束，如何修改？

9.2.3 fread 与 fwrite

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);

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，是要输出数据的地址（均指起始地址）。

size：要读写的字节数。

count：要进行读写多少个 size 字节的数据项。

fp：文件型指针。

fread 函数的返回值是读取的内容数量，fwrite 写成功后返回值是已写的对象的数量。

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

函数功能：

改变文件的位置指针。

函数调用形式：

fseek(文件类型指针，位移量，起始点)

起始点：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：以起始点为基点，向前移动的字节数。一般要求为 long 型。

fseek()成功时返回 0,失败时返回非零

下面来看代码实例【例 9.2.3-1】，通过实例我们可以更加清晰准确理解：

【例 9.2.3-1】fread 与 fwrite 以及 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;
```

```
    }
    //往文件中写入整型数
    //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个字节
    ret=fread(buf,sizeof(char),sizeof(buf)-1,fp);
    puts(buf);//打印buf的内容
    fclose(fp);
    system("pause");
}
```

fread 和 fwrite 对文件进行读写，既可以是**文本模式，也可以是二进制模式**。以“r+”，也就是文本模式打开进行读写时，往文件内写入的是字符串，写入完毕后，把 buf 清空，这时文件位置指针指向 12 个字节的位置，我们要从文件头部读取，必须通过 fseek 偏移到文件头，我们采用以当前位置为基准，向前偏移 12 个字节，接着通过 fread 读取文件，读取内容后，打印，为什么写进去的是 hello\nworld 总计 11 个字节，而想偏移到文件开头，却需要偏移 12 个字节呢，因为在文本模式下，往 txt 文件中写入\n，实际存入磁盘的是\r\n，因为所有的接口是调用的 Windows 的系统调用，这是 Windows 的底层实现，当然也不用担心，通过文本模式写入，我们以文本模式读出时，当遇到\r\n，底层接口会自动转换为\n，因此我们用 fread 再次读取出来时，得到的依然是 hello\nworld，总记 11 个字节。

如果把 fopen 的“r+”，改为“rb+”，也就是**二进制模式**，那么我们往磁盘写 11 个字节，磁盘实际存储的就是 11 个字节，这时我们往前偏移时，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 个字节，二进制模式下，如图 9.2.3-2 所示，文件大小是 11 个字节，如果这时我们自己手动双击打开，会发现没有换行 helloworld 是连在一起的，没有换行效果，原因是 txt 编辑器的换行效果是必须遇到\r\n 时才有换行效果。

到这里相信各位同学已经理解了文本模式和二进制模式的差异，文本模式，写入\n，磁盘存储的是\r\n，当然读取时，会把\r\n 以\n 形式读取出来，而二进制模式，写入\n，磁盘存储的是\n，其他方面没有差异。如何避免使用出错呢？如果以文本方式写入的内容，一定要以文本方式来读，以二进制方式写入的内容，一定要以二进制的方式来读，不能混用！

图 9.2.3-2

注释部分是二进制模式的演示，二进制模式时，我们需要以“rb+”打开文件，二进制模式，内存是什么，写进文件就是什么，保持一致，比如我们写入整形变量 i，其值为 12345，内存存储为 4 个字节，0x000004D2，写入内存也是 4 个字节，这时我们双击打开文件，看到的是乱码，读取时也需要用一个整型变量来存储，**在实际工作中，往往以二进制方式存储数据**。

思考题：如果代码中的 fseek(fp,-12,SEEK_CUR)改为 fseek(fp,-11,SEEK_CUR)，那么读取后，puts(buf)打印的效果是怎样的？

思考题：如果用 fseek 偏移位置指针到文件头，文件尾呢？

9.2.4 fgets 与 fputs

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

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，读到文件结尾返回 NULL。

```
int fputs( const char *str, 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.

下面来看代码实例【例 9.2.4-1】：

【例 9.2.4-1】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;
}
```

通过fgets，我们可以一次读取文件的一行，这样可以轻松统计文件的行数，同时拿出一行字符串以后,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OnlineJudge题目时，给fgets的buf不能过小，否则没有读取到\n时，因为到达了buf的最后，这时行数统计就会发生出错。fputs是往文件中写一个字符串，并不会额外写一个\n，比较简单，各位同学可以自己试验一下。

思考题：上面代码中fgets(buf,sizeof(buf),fp),我们没有对buf进行清空，是否有影响呢？

9.2.5 ftell

```
#include <stdio.h>
long ftell( FILE *stream );
```

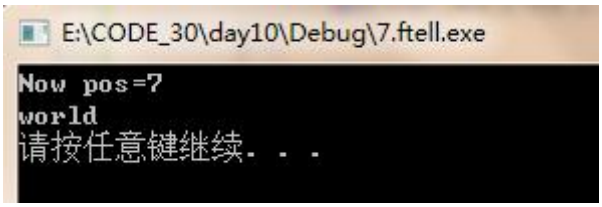
ftell()函数返回 stream(流)当前的文件位置,如果发生错误返回-1. 当我们想知道位置指针距离文件开头的位置时，就需要用 ftell 告诉我们。请看下面实例【例 9.2.5-1】：

【例 9.2.5-1】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");
}
```

最终打印结果如下：



我们往文件中写入了 hello\nworld，因为是文本模式，所以总计 12 个字节，这时通过 fseek 向前偏移 5 个字节后，通过 ftell 得到的位置指针距离文件开头的位置即为 7，这时我们再次用 fread 读取文件内容，得到的是 world。

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

9.2.6 fprintf 与 fscanf

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

fprintf()函数根据指定的 format(格式)(格式)发送信息(参数)到由 stream(流)指定的文件。fprintf()只能和 printf()一样工作（可以发现 fprintf 除去第一个形参，其他的参数与 printf 一样），printf 是将不同类型的数据以字符串的形式输出到屏幕上，fprintf 是将这些数据写入到对应的文件中，fprintf()的返回值是输出的字符数，发生错误时返回一个负值。

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

scanf 是把屏幕上我们输入的字符串数据依次格式化到各个变量中，函数 fscanf()以

scanf()的执行方式从给出的文件流中读取数据。fscanf()的返回值是事实上已赋值的变量的数目，与 scanf 等价，如果未进行任何分配时返回 EOF.

下面来看代码实例【例 9.2.6-1】：

【例 9.2.6-1】fprintf 与 fscanf 的使用

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct
{
    int num;
    char name[20];
    float score;
}stu;

int main(int argc,char* argv[])
{
    stu s={10001,"hanmeimei",95.03};
    FILE* fp;
    int ret;
    if(argc!=2)
    {
        printf("error args\n");
        system("pause");
        return -1;
    }
    fp=fopen(argv[1],"r+");
    if(NULL==fp)
    {
        perror("fopen");
    }
    fprintf(fp,"%d %s %5.2f\n",s.num,s.name,s.score);
    memset(&s,0,sizeof(s));
    fseek(fp,0,SEEK_SET);//位置指针重新定位到文件开头
    ret=fscanf(fp,"%d%s%f",&s.num,s.name,&s.score);//fscanf的返回值时匹配成功的元素个数
    printf("%d %s %5.2f\n",s.num,s.name,s.score);
    fclose(fp);
    system("pause");
    return 0;
}
```

首先 fprintf 将结构体内的数据格式化后，以字符串输出到文件中，然后我们通过 memset 清空结构体，清空后，将文件位置指针偏移到头部，然后通过 fscanf 读取文件内数据到结构体中并打印。

注意：

用 fprintf 和 fscanf 函数对磁盘文件读写，使用方便，容易理解，但由于在读取时要将 ASCII 码转换为二进制形式，在写入文件时又要将二进制形式转换成字符，花费时间比较多。因此，在内存与磁盘频繁交换数据的情况下，最好不用 fprintf 和 fscanf 函数，而用 fread 和 fwrite 函数。作业题之所以要求用 fprintf 与 fscanf，原因大家是初学者，需要所见即所得，只有将数据以字符串形式写入文件，双击打开才可以看到，如果直接讲一个整型数的内存以二进制方式写入文件，那么打开看到的是乱码。