

---

## 文件?

计算机主存 ——> 文件(硬盘)、数据库。持久性保存数据，相较于易失性存储器，我们考虑数据持久化的问题，一般数据持久化的方法有，把数据存放在磁盘文件、存放到数据库等方式。

- 程序文件  
包括源程序文件（后缀为.c），目标文件（windows环境后缀为.obj），可执行程序（windows环境后缀为.exe）。
- 数据文件  
文件的内容不一定是程序，而是程序运行时读写的数据，比如程序运行需要从中读取数据的文件，或者输出内容的文件。

---

## ★ 文件名

一个文件要有一个唯一的文件标识，以便用户识别和引用。

文件名包含3部分：`文件路径+文件名主干+文件后缀`

例如：`c:\code\test.txt`

为了方便起见，文件标识常被称为文件名。

---

## 文件操作

---

### ★ 文件指针

缓冲文件系统中，关键的概念是“文件类型指针”，简称“文件指针”。

每个被使用的文件都在内存中开辟了一个相应的**文件信息区**，用来存放文件的相关信息（如文件的名字，文件状态及文件当前的位置等）。这些信息是保存在一个结构体变量中的。该结构体类型是有系统声明的，取名FILE。

例如，VS2013编译环境提供的 `stdio.h` 头文件中有以下的文件类型申明：

比如：

```

struct _iobuf {
    char *_ptr;
    int _cnt;
    char *_base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char *_tmpfname;
};
typedef struct _iobuf FILE;

```

不同的C编译器的FILE类型包含的内容不完全相同，但是大同小异。

每当打开一个文件的时候，系统会根据文件的情况自动创建一个FILE结构的变量，并填充其中的信息，使用者不必关心细节。

**一般都是通过一个FILE的指针来维护这个FILE结构的变量，这样使用起来更加方便。**

创建一个FILE\*的指针变量:

```
FILE * pf; //文件指针变量
```

**定义pf是一个指向FILE类型数据的指针变量。**

可以使pf指向某个文件的文件信息区（是一个结构体变量）。通过该文件信息区中的信息就能够访问该文件。也就是说，**通过文件指针变量能够找到与它关联的文件。**

## ★文件的打开和关闭

ANSI C 规定使用fopen函数来打开文件，fclose来关闭文件。

函数定义如下：

```

//打开文件
FILE * fopen ( const char * filename, const char * mode );
//关闭文件
int fclose ( FILE * stream );

```

```

int main(){
    //打开名为test.txt的文件，打开方式w为只读
    FILE* pf = fopen("test.txt", "w"); //接收返回的文件信息区指针（如果文件不存在，会自动创建）
    //如果文件打开失败（例如文件不存在），会返回NULL
    if (pf == NULL) {
        perror("fopen\n");
        return 1;
    }
    //关闭文件
    fclose(pf);
    //还需要将指针置空
    pf = NULL;
    return 0;
}

```

各种文件打开方式如下：

使用方式	含义	如果指定文件不存在
“r”（只读）	为了输入数据，打开一个已经存在的文本文件	出错
“w”（只写）	为了输出数据，打开一个文本文件	建立一个新的文件
“a”（追加）	向文本文件尾添加数据	建立一个新的文件
“rb”（只读）	为了输入数据，打开一个二进制文件	出错
“wb”（只写）	为了输出数据，打开一个二进制文件	建立一个新的文件
“ab”（追加）	向一个二进制文件尾添加数据	出错
“r+”（读写）	为了读和写，打开一个文本文件	出错
“w+”（读写）	为了读和写，建议一个新的文件	建立一个新的文件
“a+”（读写）	打开一个文件，在文件尾进行读写	建立一个新的文件
“rb+”（读写）	为了读和写打开一个二进制文件	出错
“wb+”（读写）	为了读和写，新建一个新的二进制文件	建立一个新的文件
“ab+”（读写）	打开一个二进制文件，在文件尾进行读和写	建立一个新的文件

## ★文件的顺序读写

主要函数如下：

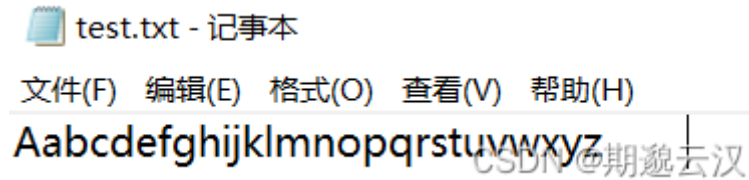
功能	函数名	适用于
字符输入函数	fgetc	所有输入流
字符输出函数	fputc	所有输出流
文本行输入函数	fgets	所有输入流
文本行输出函数	fputs	所有输出流
格式化输入函数	fscanf	所有输入流
格式化输出函数	fprintf	所有输出流
二进制输入	fread	文件
二进制输出	fwrite	文件

### 字符输出输入 fputc(), fgetc()

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

```
//写文件
fputc('A', pf); //写入单个字符A到test.txt
//循环写入
for (int i = 0; i < 26; i++) {
    fputc('a' + i, pf);
}
```

写入结果如下：

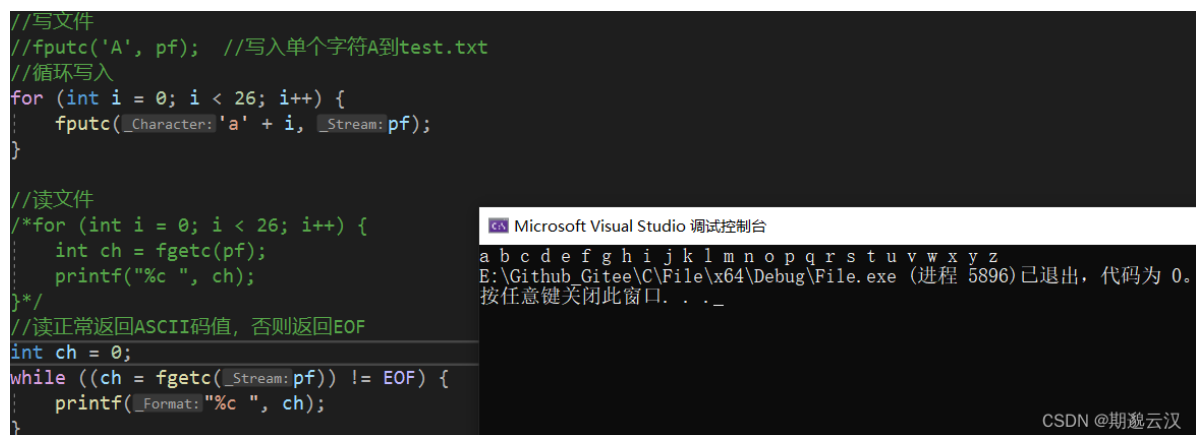


读取文件：

```
//函数定义
int fgetc ( FILE * stream );
```

读到的字符将会被返回（ASCII值）

```
//读文件
/*for (int i = 0; i < 26; i++) {
    int ch = fgetc(pf);
    printf("%c ", ch);
}*/
//读正常返回ASCII码值，否则返回EOF
int ch = 0;
while ((ch = fgetc(pf)) != EOF) {
    printf("%c ", ch);
}
```




## 按行输入输出 fputs(), fgets()

按行写入文本：

```
//函数定义
int fputs ( const char * str, FILE * stream );
```

```
int main() {
    //按行输入和输出
    FILE* pf = fopen("test.txt", "w");
    //写入时会清空原有内容
    fputs("hello\n", pf);
    fputs("world\n", pf);
    //一行一行写
}
```

 test.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

hello  
world

CSDN @期懿云汉

按行读取文件:

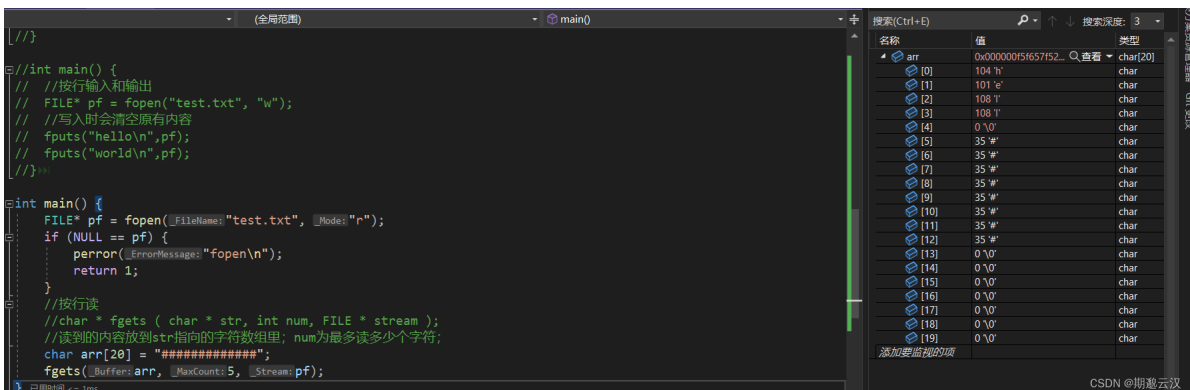
```
//函数定义
char * fgets ( char * str, int num, FILE * stream );
```

在第一行读取5个字符:

```
int main() {
    FILE* pf = fopen("test.txt", "r");
    if (NULL == pf) {
        perror("fopen\n");
        return 1;
    }
    //按行读
    //char * fgets ( char * str, int num, FILE * stream );
    //读到的内容放到str指向的字符数组里; num为最多读多少个字符;
    char arr[20] = "#####";
    fgets(arr, 5, pf); //实际读到4个

    printf("%s", arr);
}
```

读取一行后会自动放入一个 '\0'



The screenshot shows a code editor with the following C code:

```
//int main() {
//    //按行输入和输出
//    FILE* pf = fopen("test.txt", "w");
//    //写入时会清空原有内容
//    fputs("hello\n", pf);
//    fputs("world\n", pf);
//}

int main() {
    FILE* pf = fopen(_FileNames "test.txt", _Mode "r");
    if (NULL == pf) {
        perror(_ErrorMessage "fopen\n");
        return 1;
    }
    //按行读
    //char * fgets ( char * str, int num, FILE * stream );
    //读到的内容放到str指向的字符数组里; num为最多读多少个字符;
    char arr[20] = "#####";
    fgets(_Buffer arr, _MaxCount 5, _Stream pf);
}
```

On the right, a memory dump window shows the contents of the 'arr' array:

名称	值	类型
arr	0x00000015f657f52...	char[20]
[0]	104 'h'	char
[1]	101 'e'	char
[2]	108 'l'	char
[3]	108 'l'	char
[4]	0 '\0'	char
[5]	35 '#'	char
[6]	35 '#'	char
[7]	35 '#'	char
[8]	35 '#'	char
[9]	35 '#'	char
[10]	35 '#'	char
[11]	35 '#'	char
[12]	35 '#'	char
[13]	0 '\0'	char
[14]	0 '\0'	char
[15]	0 '\0'	char
[16]	0 '\0'	char
[17]	0 '\0'	char
[18]	0 '\0'	char
[19]	0 '\0'	char

读取结果:

```
hell
E:\Github_Gitee\C\File\x64\Debug\File.exe (进程 18372) 已退出，代码为 0。
按任意键关闭此窗口。 . . _
```

CSDN @期邈云汉

在第一行读取10个字符:

```
int main() {
    FILE* pf = fopen("test.txt", "r");
    if (NULL == pf) {
        perror("fopen\n");
        return 1;
    }
    //按行读
    //char * fgets ( char * str, int num, FILE * stream );
    //读到的内容放到str指向的字符数组里; num为最多读多少个字符;
    char arr[20] = "#####";
    fgets(arr, 10, pf); //\n也会读到

    printf("%s", arr);
}
```

搜索(Ctrl+E) 🔍 ↑ ↓ 搜索深度: 3

名称	值	类型
arr	0x00000067505afb... 🔍 查看	char[20]
[0]	104 'h'	char
[1]	101 'e'	char
[2]	108 'l'	char
[3]	108 'l'	char
[4]	111 'o'	char
[5]	10 '\n'	char
[6]	0 '\0'	char
[7]	35 '#'	char
[8]	35 '#'	char
[9]	35 '#'	char
[10]	35 '#'	char
[11]	35 '#'	char
[12]	35 '#'	char
[13]	0 '\0'	char
[14]	0 '\0'	char
[15]	0 '\0'	char
[16]	0 '\0'	char
[17]	0 '\0'	char
[18]	0 '\0'	char
[19]	0 '\0'	char

添加要监视的项 CSDN @期邈云汉

\n 会读取到，打印结果如下:

hello

E:\Github\_Gitee\C\File\x64\Debug\File.exe (进程 15532) 已退出，代码为 0。  
按任意键关闭此窗口。 . . \_

CSDN @期邈云汉

把这两行都读取如下：

//读到的内容放到str指向的字符数组里；num为最多读多少个字符；

```
char arr[20] = "#####";
fgets(arr, 10, pf); //\n也会读到
printf("%s", arr);
```

```
fgets(arr, 10, pf); //\n也会读到
printf("%s", arr);
```

```
//按行读
//char * fgets ( char * str, int num, FILE * stream );
//读到的内容放到str指向的字符数组里；num为最多读多少个字符；
char arr[20] = "#####";
fgets(_Buffer:arr, _MaxCount:10, _Stream:pf); //\n也会读到
printf(_Format:"%s", arr);

fgets(_Buffer:arr, _MaxCount:10, _Stream:pf); //\n也会读到
printf(_Format:"%s", arr);
```

C:\ Microsoft Visual Studio 调试控制台

hello  
world

E:\Github\_Gitee\C\File\x64\Debug\File.exe (进程 16748) 已退出，代码为 0。  
按任意键关闭此窗口。 . . \_


CSDN @期邈云汉

## 读取结构体

//读取结构体

```
struct Stu {
    char name[10];
    int age;
    float height;
};
int main() {
    struct Stu s = { "张三", 18, 177.6f };

    FILE* pf = fopen("test.txt", "w");
    if (NULL == pf) {
        perror("fopen\n");
        return 1;
    }
    //进行写文件
    fprintf(pf, "%s %d %f", s.name, s.age, s.height);
}
```

 test.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

张三 18 177.699997

CSDN @期邈云汉

## 读取文件:

```
//读取数据 int fscanf ( FILE * stream, const char * format, ... );
//从文件流里读取
fscanf(pf, "%s %d %f", s.name, &(s.age), &(s.height));

printf("%s %d %f\n", s.name, s.age, s.height);
```

```
//读取数据 int fscanf ( FILE * stream, const char * format, ... );
//从文件流里读取
fscanf(_Stream: pf, _Format: "%s %d %f", s.name, &(s.age), &(s.height));
printf(_Format: "%s %d %f\n", s.name, s.age, s.height);
```

Microsoft Visual Studio 调试控制台  
张三 18 177.699997  
E:\Github\_Gitee\C\File\x64\Debug\File.exe (进程 13600) 已退出, 代码为 0。  
按任意键关闭此窗口

输入流 (读) ----

输出流 (写) ----

对任何一个C程序, 运行起来默认打开三个流:

stdin--标准输入流 > 键盘  
stdout--标准输出流 > 屏幕  
stderr--标准错误流 > 屏幕

```
int main() {
    int ch = fgetc(stdin); //从键盘上读取
    fputc(ch, stdout); //输出到屏幕上

    return 0;
}
```

```
int main() {
    int ch = fgetc(_Stream: stdin); //从键盘上读取
    fputc(_Character: ch, _Stream: stdout); //输出到屏幕上

    return 0;
}
```

Microsoft Visual Studio 调试控制台

A  
A  
E:\Github\_Gitee\C\File\x64\Debug\File.exe (进程 9496) 已退出, 代码为 0。  
按任意键关闭此窗口. . .

CSDN @期邈云汉

## 比较printf(), fprintf(); scanf(), fscanf()

```
int printf ( const char * format, ... );
```

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

```
int scanf ( const char * format, ... );
```

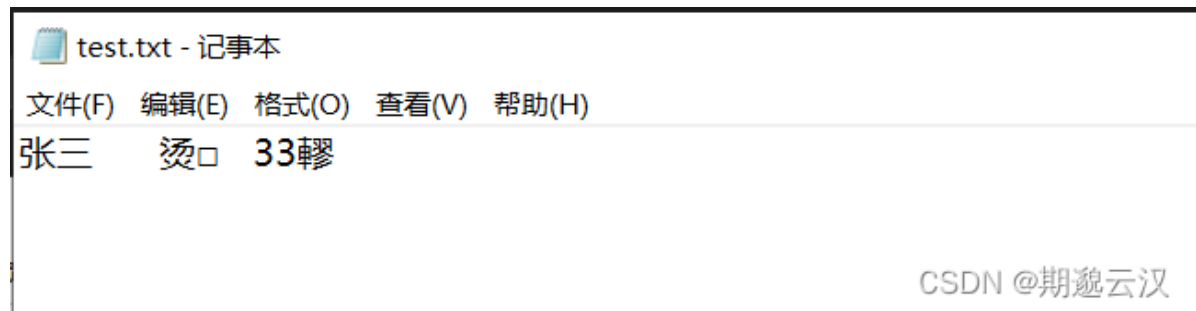
```
int fscanf ( FILE * stream, const char * format, ... );
```



相当于文件输入输出流的指定，我们常用的printf，scanf，就是将输出流和输入流分别指定为stdout，stdin。

## 二进制形式读写

```
//二进制写入
int main() {
    struct Stu s = { "张三", 20, 111.1 };
    FILE* pf = fopen("test.txt", "wb"); //二进制写入
    if (NULL == pf) {
        perror("fopen\n");
        return 1;
    }
    //size_t fwrite ( const void * ptr, size_t size, size_t count, FILE * stream
);
    //从ptr指针里取count 个大小为size的数据到文件流里
    fwrite(&s, sizeof(s), 1, pf);
}
```



### 二进制读取

```
int main() {
    struct Stu s = { 0 };
    FILE* pf = fopen("test.txt", "rb"); //二进制读取
    if (NULL == pf) {
        perror("fopen\n");
        return 1;
    }
    //size_t fread ( void * ptr, size_t size, size_t count, FILE * stream )
    //从文件流里读取count个大小为size的数据到ptr所指向的空间去
    fread(&s, sizeof(s), 1, pf);

    printf("%s %d %f\n", s.name, s.age, s.height);
    return 0;
}
```

```
int main() {
    struct Stu s = { 0 };
    FILE* pf = fopen(_FileName: "test.txt", _Mode: "rb"); //二进制读取
    if (NULL == pf) {
        perror(_ErrorMessage: "fopen\n");
        return 1;
    }
    //size_t fread ( void * ptr, size_t size, size_t count, FILE * stream )
    //从文件流里读取count个大小为size的数据到ptr所指向的空间去
    fread(_Buffer: &s, _ElementSize: sizeof(s), _ElementCount: 1, _Stream: pf);

    printf(_Format: "%s %d %f\n", s.name, s.age, s.height);
    return 0;
}

```

Microsoft Visual Studio 调试控制台

张三 20 111.099998

E:\Github\_Gitee\C\File\x64\Debug\File.exe (进程 2624) 已退出，代码为 0。  
按任意键关闭此窗口。 . . .

CSDN @期邈云汉

## ★ 对比三种函数

适用于标准输入/输出流的格式化输入/输出语句

scanf  
printf

适用于所有的输入或输出流的格式化输入/输出语句

fscanf 按照一定的格式从输入流（文件、stdin）输入数据  
fprintf 按照一定格式向输出流（文件、stdout）输出数据

**sprintf**:把格式化的数据按照一定格式转化成字符串

```
int sprintf ( char * str, const char * format, ... );
```

```
int main() {
    struct Stu s = { "张三", 10, 12.8f };

    char buf[40] = { 0 };
    sprintf(buf, "%s %d %f", s.name, s.age, s.height);

    printf("%s", buf);
    return 0;
}

```

**sscanf**: 从字符串中按照一定格式读取格式化数据

```
int sscanf ( const char * s, const char * format, ... );
```

```
int main() {
    struct Stu s = { "张三", 10, 12.8f };

```

```

    struct Stu tmp = { 0 };

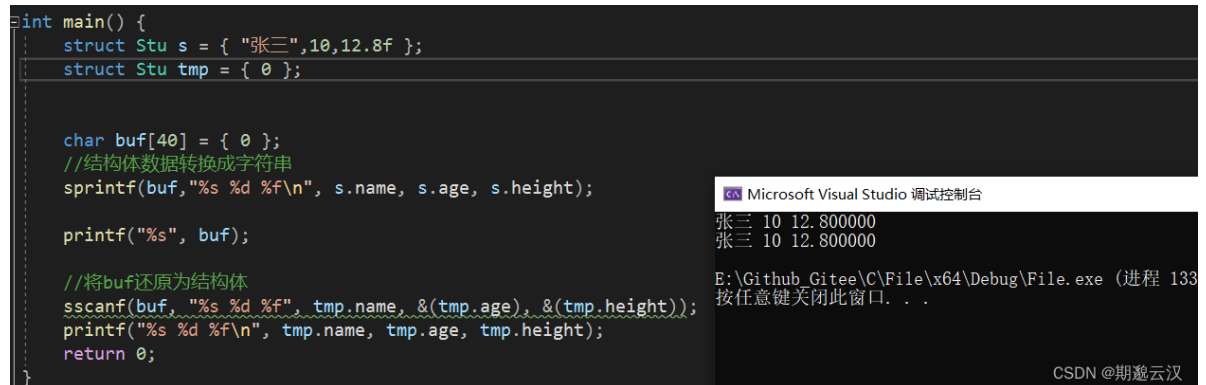
    char buf[40] = { 0 };
    //结构体数据转换成字符串
    sprintf(buf, "%s %d %f\n", s.name, s.age, s.height);

    printf("%s", buf);

    //将buf还原为结构体
    sscanf(buf, "%s %d %f", tmp.name, &(tmp.age), &(tmp.height));
    printf("%s %d %f\n", tmp.name, tmp.age, tmp.height);
    return 0;
}

```

打印结果验证:



```

int main() {
    struct Stu s = { "张三", 10, 12.8f };
    struct Stu tmp = { 0 };

    char buf[40] = { 0 };
    //结构体数据转换成字符串
    sprintf(buf, "%s %d %f\n", s.name, s.age, s.height);

    printf("%s", buf);

    //将buf还原为结构体
    sscanf(buf, "%s %d %f", tmp.name, &(tmp.age), &(tmp.height));
    printf("%s %d %f\n", tmp.name, tmp.age, tmp.height);
    return 0;
}

```

Microsoft Visual Studio 调试控制台

```

张三 10 12.800000
张三 10 12.800000
E:\Github_Gitee\C\File\x64\Debug\File.exe (进程 133)
按任意键关闭此窗口. . .

```

CSDN @期邈云汉

到这里, 考虑改造文件操作的简单通讯录实现。

也就是退出通讯录后, 数据不能丢, 下一次重新运行, 还可以看到上一次保存的信息。

如何保存?

如何加载?

(参考通讯录实现教程)

[简单通讯录实现](#)

## ★ 文件的随机读写

### fseek

根据文件指针的位置和偏移量来定位文件指针

```

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

```

```


int main() {
    FILE* pf = fopen("test.txt", "r"); //abcfgh
    if (pf == NULL) {
        perror("fopen\n");
        return 1;
    }
    //随机读文件
    fseek(pf, 4, SEEK_SET);
    int tmp = fgetc(pf);
    printf("%c\n", tmp);
}

```

```
//获取当前偏移量
printf("%d\n",ftell(pf));

//rewind让pf回到起始位置
rewind(pf);
printf("%d\n", ftell(pf));

return 0;
}
```

 Microsoft Visual Studio 调试控制台

```
g
5
0

E:\Github_Gitee\C\File\x64\Debug\File.exe (进程 16604) 已退出，代码为 0。
按任意键关闭此窗口。 . . .

CSDN @期邈云汉
```

- **ftell**

返回文件指针相对于起始位置的偏移量

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

- **rewind**

让文件指针的位置回到文件的起始位置

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

## 文本文件和二进制文件

根据数据的组织形式，数据文件被称为文本文件或者二进制文件。

数据在内存中以二进制的形式存储，如果不加转换的输出到外存，就是二进制文件。

如果要求在外存上以ASCII码的形式存储，则需要存储前转换。以ASCII字符的形式存储的文件就是文本文件。

一个数据在内存中是怎么存储的呢？

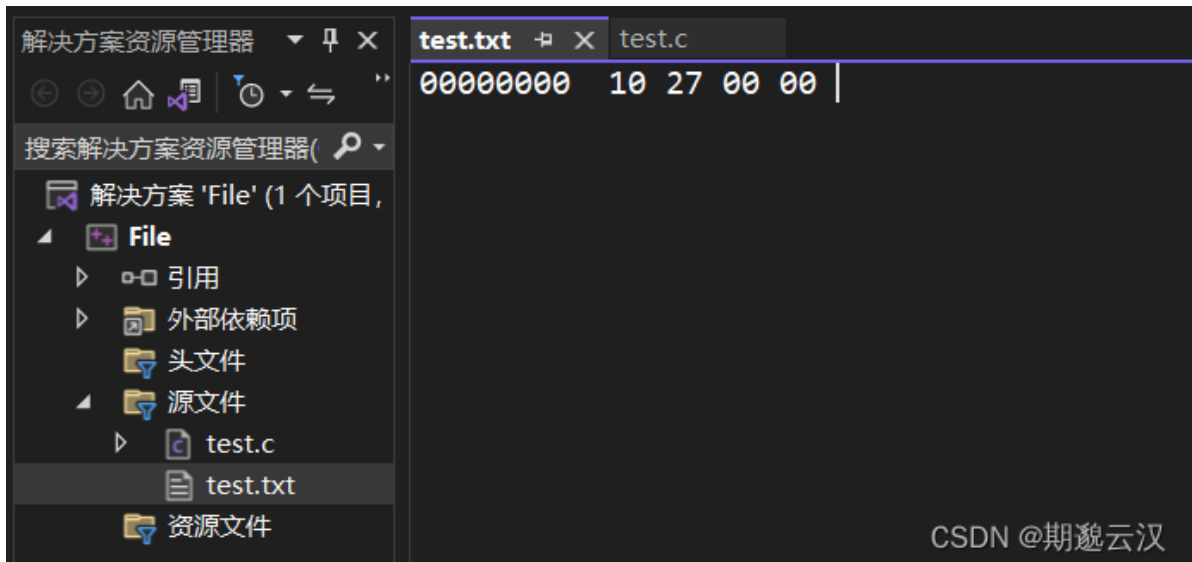
字符一律以ASCII形式存储，数值型数据既可以用ASCII形式存储，也可以使用二进制形式存储。

如有整数10000，如果以ASCII码的形式输出到磁盘，则磁盘中占用5个字节（每个字符一个字节），而二进制形式输出，即为int整形，则在磁盘上只占4个字节（VS2013测试）。

简单测试：

```
int main()
{
    int a = 10000;
    FILE* pf = fopen("test.txt", "wb");
    fwrite(&a, 4, 1, pf); //二进制的形式写到文件中
    fclose(pf);
    pf = NULL;
    return 0;
}
```

## 以二进制存放



10 27 00 00 就是10000的二进制码（写为十六进制形式），这里test.txt即为二进制文件。

如果采用ASCII码值进行存储，1（字符）的ASCII码为49，0（字符）为48

## ASCII形式

00110001	00110000	00110000	00110000	00110000
(1)	(0)	(0)	(0)	(0)

00000000	00000000	00100111	00010000
----------	----------	----------	----------

## 二进制形式存储

CSDN @期邈云汉

## 文件读取结束的判定

feof

在文件读取过程中，不能用feof函数的返回值直接用来判断文件的是否结束。

而是应用于当文件读取结束的时候，判断是读取失败结束，还是遇到文件尾结束。

1. 文本文件读取是否结束，判断返回值是否为 EOF（fgetc），或者 NULL（fgets）

例如：

fgetc 判断是否为 EOF。

fgets 判断返回值是否为 NULL

2. 二进制文件的读取结束判断，判断返回值是否小于实际要读的个数。

例如：

fread判断返回值是否小于实际要读的个数。

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int c; // 注意: int, 非char, 要求处理EOF
    FILE* fp = fopen("test.txt", "r");
    if(!fp) {
        perror("File opening failed");
        return EXIT_FAILURE;
    }
    //fgetc 当读取失败的时候或者遇到文件结束的时候，都会返回EOF
    while ((c = fgetc(fp)) != EOF) // 标准C I/O读取文件循环
    {
        putchar(c);
    }

    //判断是什么原因结束的
    if (ferror(fp))
        puts("I/O error when reading");
    else if (feof(fp))
        puts("End of file reached successfully");
    fclose(fp);
}
```

## 文件缓冲区

ANSI C 标准采用“缓冲文件系统”处理的数据文件的，所谓缓冲文件系统是指系统自动地在内存中为程序中每一个正在使用的文件开辟一块“文件缓冲区”。从内存向磁盘输出数据会先送到内存中的缓冲区，装满缓冲区后才一起送到磁盘上。如果从磁盘向计算机读入数据，则从磁盘文件中读取数据输入到内存缓冲区（充满缓冲区），然后再从缓冲区逐个地将数据送到程序数据区（程序变量等）。缓冲区的大小根据C编译系统决定的。

fflush(pf); 强制刷新//刷新缓冲区时，才将输出缓冲区的数据写到文件（磁盘）

```
#include <windows.h>
int main() {
    FILE* pf = fopen("test.txt", "w");
    fputs("abcdef", pf); //先将代码放在输出缓冲区

    printf("睡眠10秒-已经写数据了，打开test.txt文件，发现文件没有内容\n");
    Sleep(10000); //毫秒-->10s
    printf("刷新缓冲区\n");
    fflush(pf); //刷新缓冲区时，才将输出缓冲区的数据写到文件（磁盘）
}
```

//注: `fflush` 在高版本的VS上不能使用了

```
printf("再睡眠10秒-此时，再次打开test.txt文件，文件有内容了\n");
```

```
sleep(10000);
```

```
fclose(pf);
```

//注: `fclose`在关闭文件的时候，也会刷新缓冲区

```
pf = NULL;
```

```
return 0;
```

```
}
```

---