2.文件的打开和关闭

文件在读写之前应该先打开文件，在使用结束之后应该关闭文件。
在编写程序的时候，在打开文件的同时，都会返回一个FILE*的指针变量指向该文件，也相当于建立了指针和文件的关系。
ANSIC 规定使用fopen函数来打开文件，fclose来关闭文件。
要记住的是当打开文件后对数据进行处理完一定要关闭文件，否则可能会造成数据的丢失。

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

对于文件的写入和读取方式，重点掌握以下几种即可。

文件使用方式	含义
“r”（只读）	为了输入数据，打开一个已经存在的文本文件
“w”（只写）	为了输出数据，打开一个文本文件
“a”（追加）	向文本文件尾添加数据
“rb”（只读）	为了输入数据，打开一个二进制文件
“wb”（只写）	为了输出数据，打开一个二进制文件

实例代码：

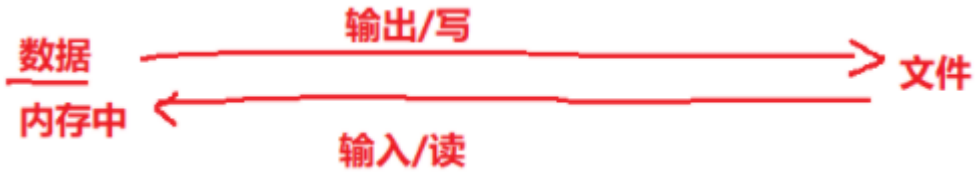
```
/* fopen fclose example */
#include <stdio.h>
int main ()
{
  FILE * pFile;
  //打开文件
  pFile = fopen ("myfile.txt","w");//以输出的形式（写）打开文件
  //文件操作
  if (pFile!=NULL)
 {
    fputs ("fopen example",pFile);//以字符串的形式写入
    //关闭文件
    fclose (pFile);
 }


```

```
    return 0;
}
```

4.文件的顺序读写

文件的输出/写入就是将数据写入到文件当中，而文件的输入/读取就是将文件中的内容读取到内存当中。



以下的对于文件的读写方式的函数均要求掌握

功能	函数名	
字符输入函数	fgetc	
字符输出函数	fputc	
文本行输入函数	fgets	
文本行输出函数	fputs	
格式化输入函数	fscanf	
格式化输出函数	fprintf	
二进制输入	fread	
二进制输出	fwrite	

四、fseek函数

根据文件指针的位置和偏移量来定位文件指针。文件指针顾名思义也是一个指针，它能指向一个字符串中的某个位置。它要接收的参数有：

fseek

Moves the file pointer to a specified location.

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

Function	Required Header
fseek	<stdio.h>

第一个参数是文件指针的名字（流），第二个参数是文件指针向后偏移数，第三个参数是fseek函数中规定的三个选项之中的其一。

SEEK_CUR

Current position of file pointer

SEEK_END

End of file

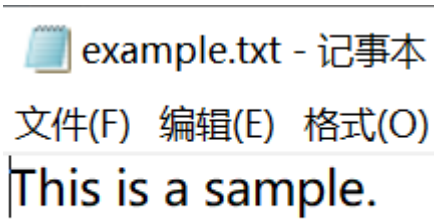
SEEK_SET

Beginning of file

这三项中第一项是SEEK_CUR，即当前文件指针的偏移处开始向后偏移。第二项是SEEK_END，即从文件的最末尾处开始向前偏移，当然在偏移数一定要为负数才能读取文件中的内容。第三项是SEEK_SET，即从文件的最前端处开始向后偏移。举个例子：

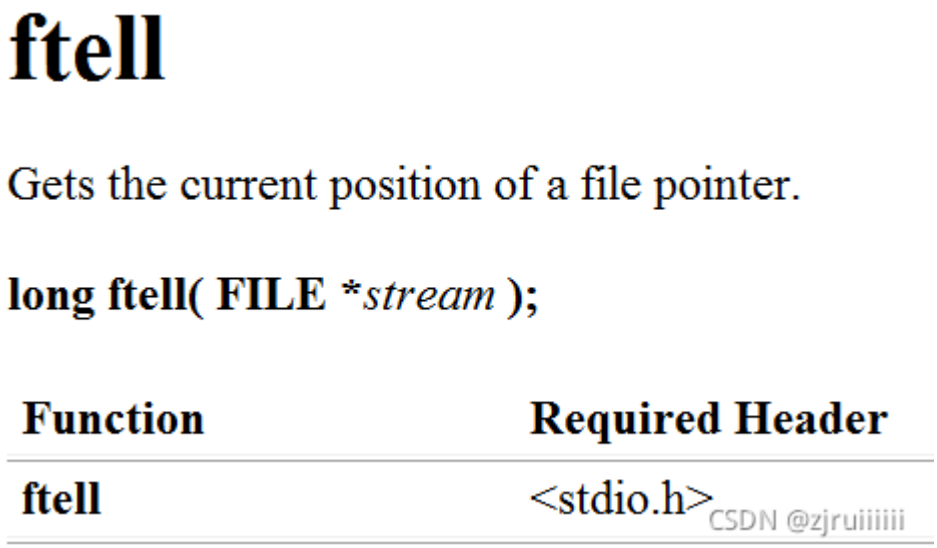
```
#include <stdio.h>
int main ()
{
  FILE * pFile;
  pFile = fopen ( "example.txt" , "wb" );
  fputs ( "This is an apple." , pFile );
  fseek ( pFile , 9 , SEEK_SET );
  fputs ( " sam" , pFile );
  fclose ( pFile );
  return 0; }
```

为什么最后在记事本中打印出的结果是This is a sample.呢?原因是在第一次fputs中是把This is an apple.先放入记事本当中，当调用fseek函数时，从当前的文件指针处向后偏移9个字节，文件指针一开始默认指向的是文件的首地址处。因此向后偏移9个字节后（偏移一个字节包括空格）指向的是最后一个空格的地址处。而第二次fputs函数是将“ sam”这个内容在上次文件指针指向的地址处开始写入。因此最后程序运行的结果如图：



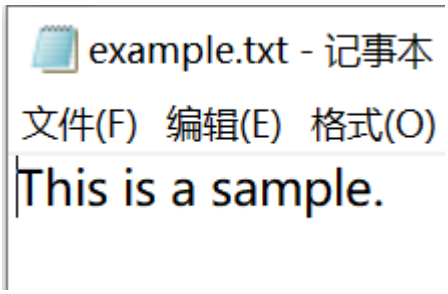
五、ftell函数

返回文件指针相对于起始位置的偏移量。

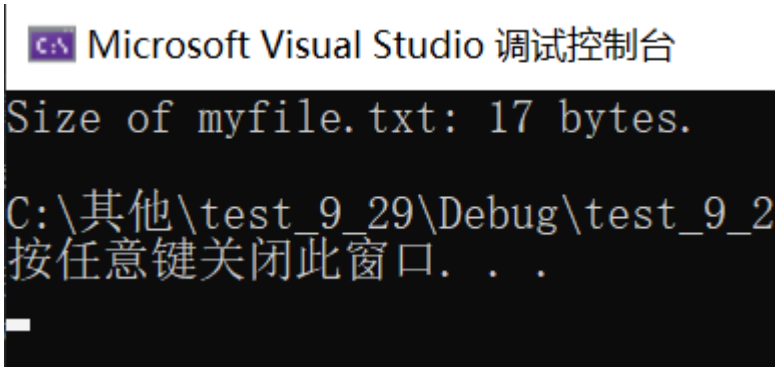


这个函数比较简单，输入的参数为文件指针流，而返回值的类型为int，即返回的是文件指针所指向的偏移量处。

```
#include <stdio.h>
int main ()
{
  FILE * pFile;
  long size;
  pFile = fopen ("myfile.txt","rb");
  if (pFile==NULL) perror ("Error opening file");
  else
  {
    fseek (pFile, 0, SEEK_END);   // non-portable
    size=ftell (pFile);
```

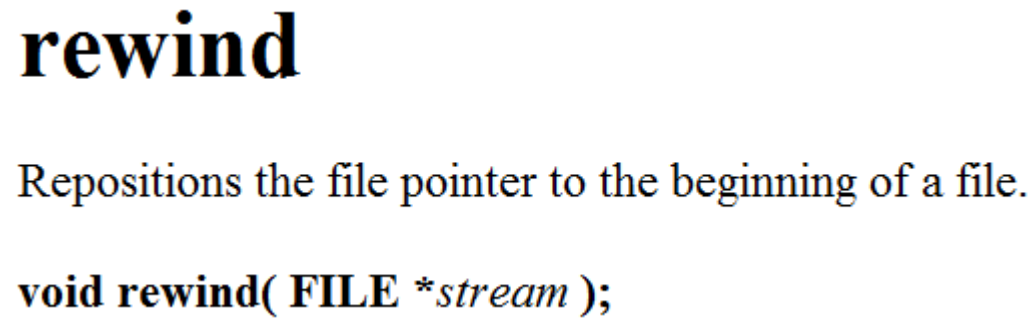


因为是从文件内容的最末尾处开始相对于起始位置的偏移量。则结果为17。
代码运行结果为：



六、rewind函数

让文件指针的位置回到文件的起始位置。



rewind

Repositions the file pointer to the beginning of a file.

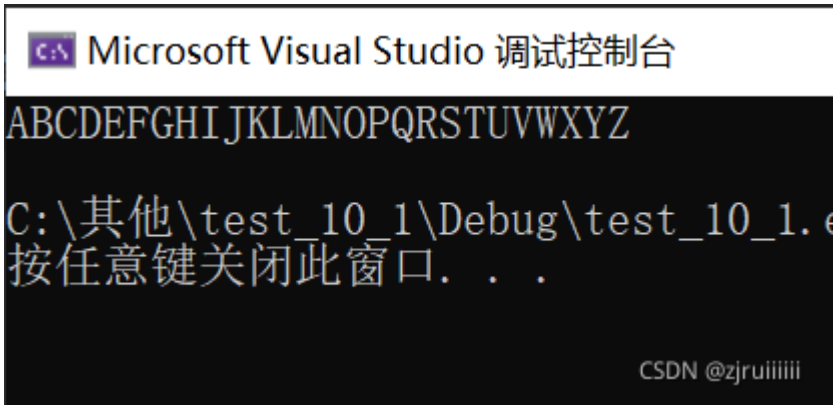
void rewind(FILE *stream);

rewind函数的返回值类型为void型，它所需要的参数是文件指针流。这个函数相对来说也比较简单，我们直接举例子。

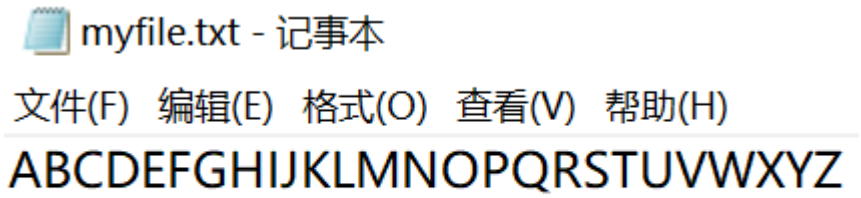
```
#include <stdio.h>
int main ()
{
  int n;
  FILE * pFile;
  char buffer [27];
  pFile = fopen ("myfile.txt","w+");
  for ( n='A' ; n<='Z' ; n++)
    fputc ( n, pFile);
  rewind (pFile);
```

```
    fread (buffer,1,26,pFile);
    fclose (pFile);
    buffer[26]='\0';
    puts (buffer);
    return 0; }
```

代码运行结果：



并且在程序的文件夹中有此内容的记事本产生：



七、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，则需要在存储前转换。以ASCII字符的形式存储的文件就是文本文件。（如整数10000，需要以ASCII码输出到磁盘上，则在磁盘中的存储形式就是10000）。

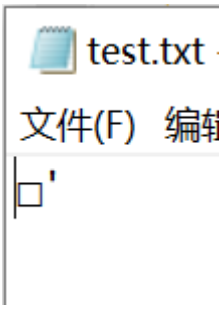
如有整数10000，如果以ASCII码的形式输出到磁盘，则磁盘中占用5个字节（每个字符一个字节），而二进制形式输出，则在磁盘上只占4个字节（VS2013测试）。

再用整数10000举例。如果以二进制的形式输出到磁盘上，则在磁盘上是以二进制的形式存储。但是我们到文件底下去看二进制形式的文本时，都是乱码无法看懂（但机器能够看懂）。此时我们再将该文本文件移到编译器（VS2019）中。而编译器内有一个二进制编辑器能够将该乱码翻译为二进制数显示出来。详细步骤如下：

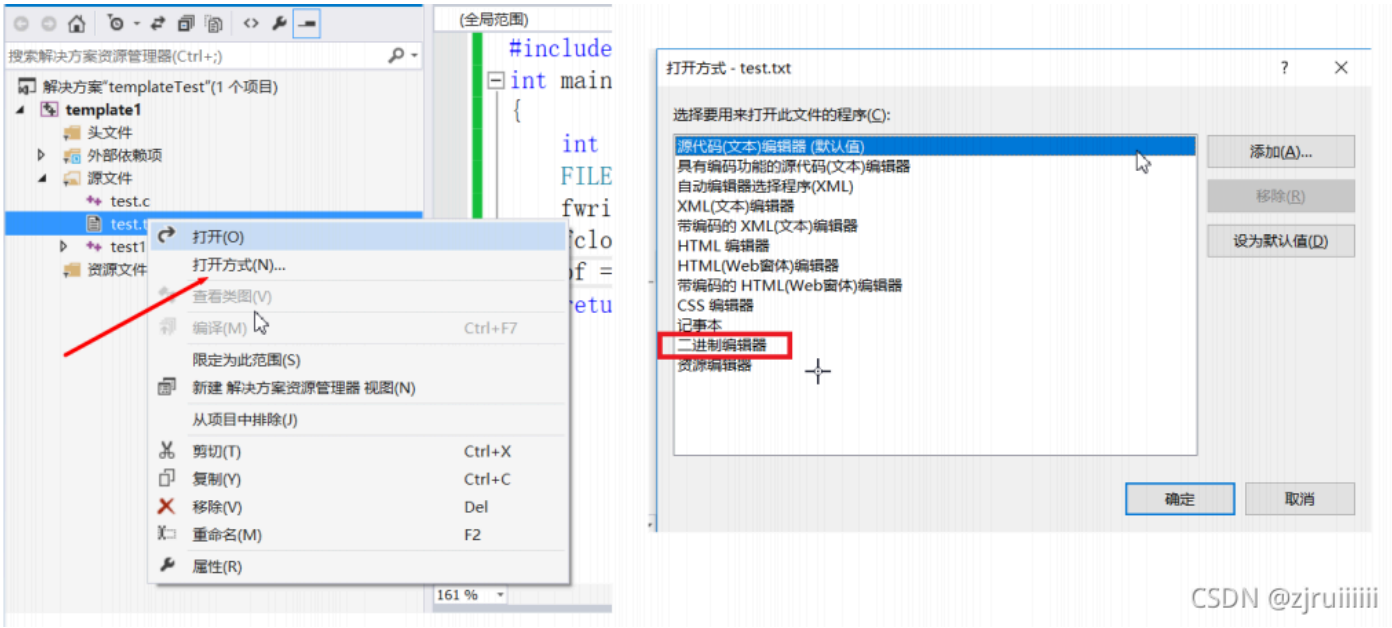
代码：

```
#include <stdio.h>
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; }
```

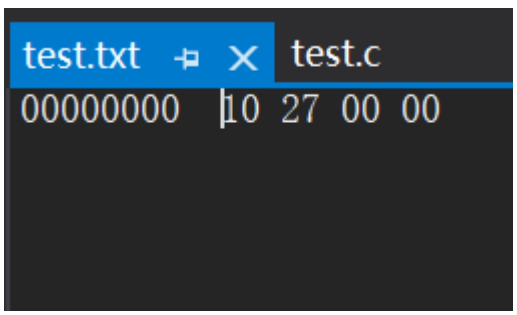
到文件底下去查看文本：



将该文本移到编译器中后按照以下图例操作：



此时我们在编译器中打开该文本：



是什么原因让10000用二进制的形式存储变为了10 27 00 00呢？原因是我们先将10000的二进制序列写出来，为： `00000000 00000000 00100111 00010000` ，每四位则为一个16进制数字。则结果为

00 00 27 10，但是我们的编译器是以小端的形式存储的。即数据的低位存储到内存的低地址中，数据的高位存储到高地址中。则存储的形式就为：10 27 00 00 。

八、文件读取结束的判定

1.feof函数的错误使用

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。 （feof函数是判断结束过程而不是判断结束的结果）

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

Return Value

fgetc and **_fgetchar** return the character read as an **int** or return **EOF** to indicate an error or end of file.

fgets 判断返回值是否为 NULL.

Return Value

Each of these functions returns *string*. **NULL** is returned to indicate an error or an end-of-file condition.

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。

文件文本中正确使用feof函数的例子：

```
#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
```

二进制文件中正确使用feof函数的例子：

```c
#include <stdio.h>
enum { SIZE = 5 };
int main(void) {
    double a[SIZE] = {1.,2.,3.,4.,5.};
    FILE *fp = fopen("test.bin", "wb"); // 必须用二进制模式
    fwrite(a, sizeof *a, SIZE, fp); // 写 double 的数组
    fclose(fp);
    double b[SIZE];
    fp = fopen("test.bin","rb");
    size_t ret_code = fread(b, sizeof *b, SIZE, fp); // 读 double 的数组
    if(ret_code == SIZE) {
        puts("Array read successfully, contents: ");
        for(int n = 0; n < SIZE; ++n) printf("%f ", b[n]);
        putchar('\n');
   } else { // error handling
       if (feof(fp))
          printf("Error reading test.bin: unexpected end of file\n");
       else if (ferror(fp)) {
           perror("Error reading test.bin");
       }
   }
    fclose(fp);
}
```

九、文件缓冲区

说到文件缓冲区，我们就自然而然想到输入缓冲区，即当一个字符一个字符从键盘上输入时，并不是直接输入到磁盘内，而是先放到输入缓冲区，而当输入缓冲区内的字符放满后，文件缓冲区才向磁盘内输入字符。

文件缓冲区也是一样的道理。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。

测试代码：

```c
#include <stdio.h>
#include <windows.h>
//VS2013 WIN10环境测试
int main()
{
 FILE*pf = fopen("test.txt", "w");
 fputs("abcdef", pf);//先将代码放在输出缓冲区

```

```c
    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);
    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; }
```

我们可以测试一下这个代码，在程序第一个到fgets函数处时，立刻去打开test.txt文本文件，我们会发现里面没有内容，而我们用刷新文件缓冲区的fflush函数时再次打开test.txt文本文件时，会发现里面已经有输入的内容。则能够证实的确有文件缓冲区的存在。

因为有缓冲区的存在，C语言在操作文件的时候，需要做刷新缓冲区或者在文件操作结束的时候关闭文件。如果不做，可能导致读写文件的问题。