C语言文件操作

- 什么是文件
- 文件名
- 文件类型
- 文件缓冲区
- 文件指针
- 文件的打开和关闭
- 文件的顺序读写
- 文件的随机读写
- 文件结束的判定

正文开始@比特科技

什么是文件

磁盘上的文件是文件。

但是在程序设计中，我们一般谈的文件有两种：程序文件、数据文件

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。

本章讨论的是数据文件。

在以前各章所处理数据的输入输出都是以终端为对象的，即从终端的键盘输入数据，运行结果显示到显示器上。

其实有时候我们会把信息输出到磁盘上，当需要的时候再从磁盘上把数据读取到内存中使用，这里处理的就是磁盘上文件。

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**。

文件类型

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**。

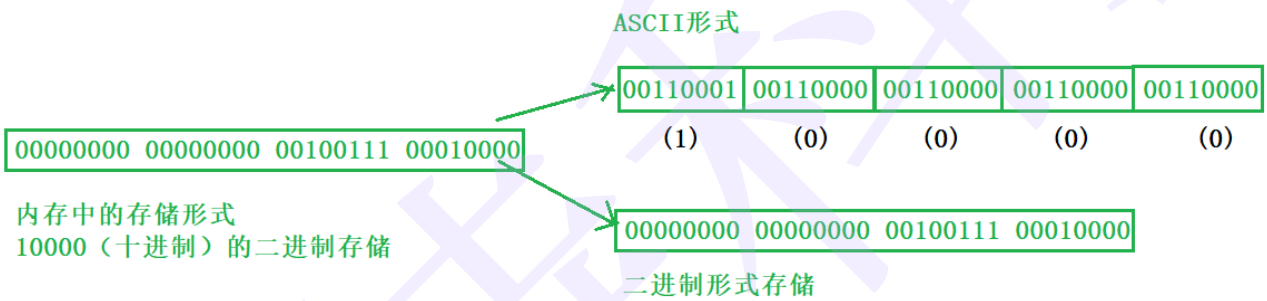
数据在内存中以二进制的形式存储，如果不加转换的输出到外存，就是**二进制文件**。

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**。

一个数据在内存中是怎么存储的呢？

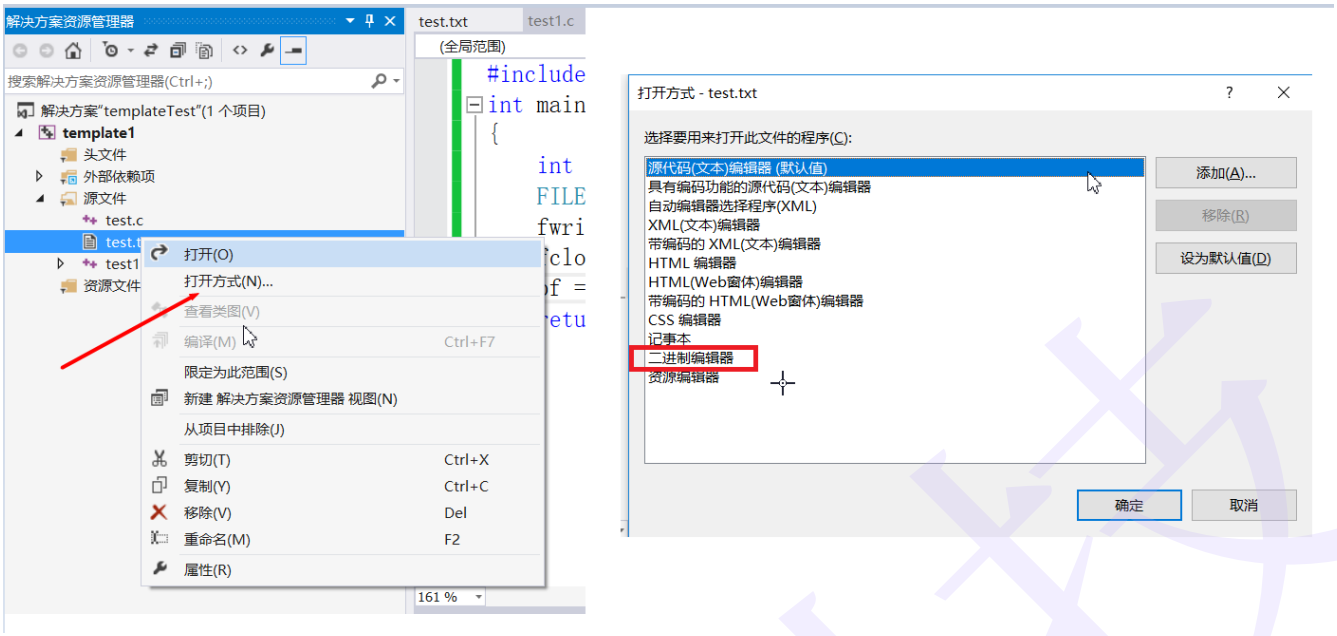
字符一律以ASCII形式存储，数值型数据既可以用ASCII形式存储，也可以使用二进制形式存储。

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，则在磁盘上只占4个字节（VS2013测试）。



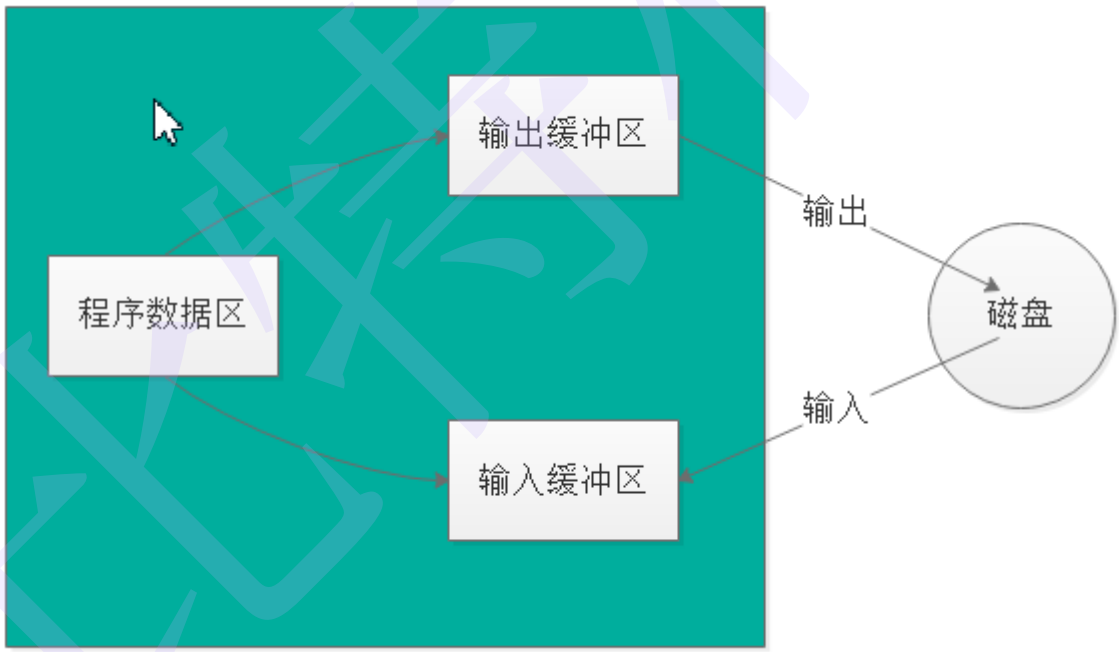
测试代码：

```c
#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;
}
```



文件缓冲区

ANSIC 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。



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,VS2008编译环境提供的`stdio.h`头文件中有以下的文件类型申明：

```
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。

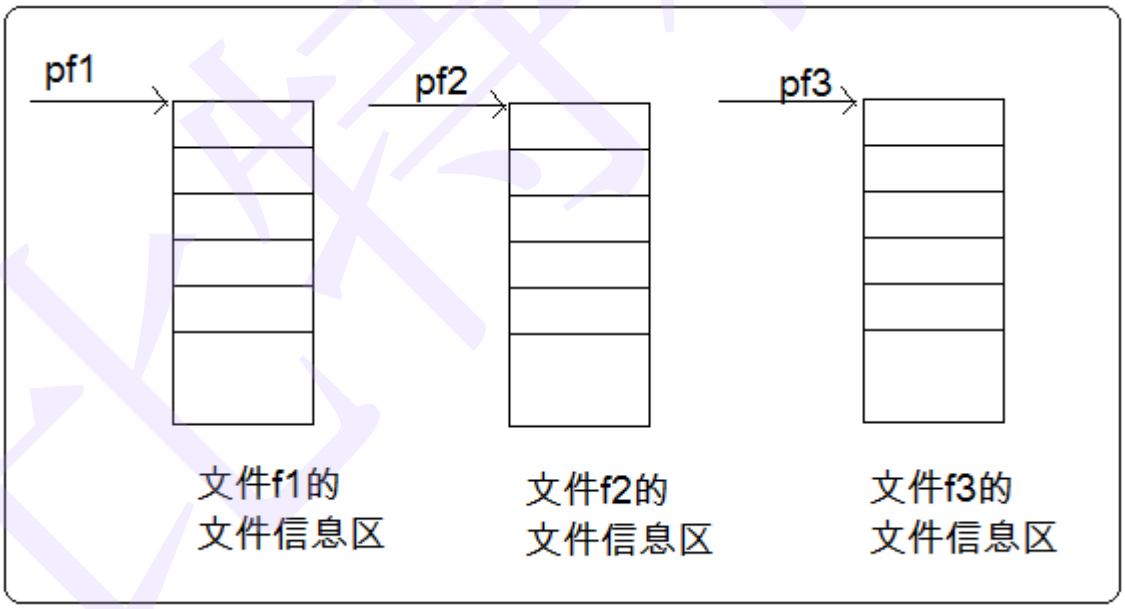
一般都是通过一个FILE*的指针来维护这个FILE结构的变量，这样使用起来更加方便。*

下面我们可以创建一个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**。

比如：



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。

```
FILE * fopen ( const char * filename, const char * mode );
int fclose ( FILE * stream );
```

打开方式如下：

文件使用方式	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	出错
“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

实例代码：

```
/* fopen fclose example */
#include <stdio.h>
int main ()
{
  FILE * pFile;
  pFile = fopen ("myfile.txt","w");
  if (pFile!=NULL)
  {
    fputs ("fopen example",pFile);
    fclose (pFile);
  }
  return 0;
}
```

文件的顺序读写

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

对比一组函数：

> scanf/fscanf/sscanf
>
> printf/fprintf/sprintf

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 );
```

例子：

```
/* fseek example */
#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;
}
```

ftell

> 返回文件指针相对于起始位置的偏移量

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

例子：

```
/* ftell example : getting size of a file */
#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);
    fclose (pFile);
    printf ("Size of myfile.txt: %ld bytes.\n",size);
  }
  return 0;
}
```

rewind

> 让文件指针的位置回到文件的起始位置

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

例子：

```
/* rewind example */
#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;
```

```
}
```

文件结束判定

被错误使用的`feof`

牢记：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。

正确的使用：

文本文件的例子：

```c
#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);
}
```

二进制文件的例子：

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

```c
enum { SIZE = 5 };
int main(void)
{
    double a[SIZE] = {1.0,2.0,3.0,4.0,5.0};
    double b = 0.0;
    size_t ret_code = 0;
    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);

    fp = fopen("test.bin","rb");
    // 读 double 的数组
    while((ret_code = fread(&b, sizeof(double), 1, fp))>=1)
    {
        printf("%lf\n",b);
    }
    if (feof(fp))
        printf("Error reading test.bin: unexpected end of file\n");
    else if (ferror(fp)) {
        perror("Error reading test.bin");
    }
    fclose(fp);
    fp = NULL;
}
```

参考链接：https://zh.cppreference.com/w/c

本章完