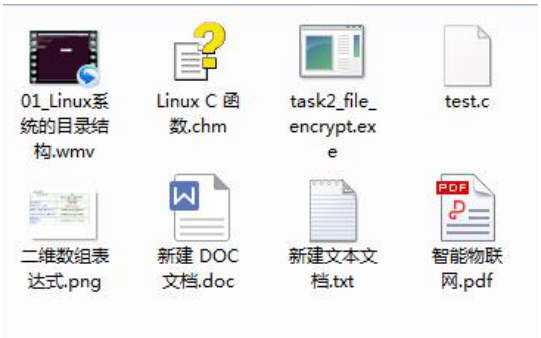
第十章 文件

1.1 文件的概念

凡是使用过文件的人对文件都不会感到陌生



文件用来存放程序、文档、音频、视频数据、图片等数据的。
文件就是存放在磁盘上的，一些数据的集合。

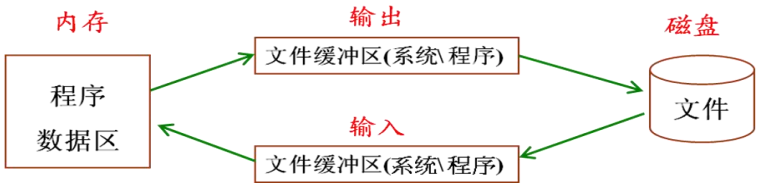
在 windows 下可以通过写字板或记事本打开文本文件对文件进行编辑保存。写字板和记事本是微软程序员写的程序，对文件进行打开、显示、读写、关闭。
作为一个程序员，必须掌握编程实现创建、写入、读取文件等操作

对文件的操作是经常要用到的知识，比如：写飞秋软件传送文件 等

1.1.1 文件的定义

磁盘文件：（我们通常认识的文件）
指一组相关数据的有序集合,通常存储在外部介质(如磁盘)上，使用时才调入内存。
设备文件：
在操作系统中把每一个与主机相连的输入、输出设备看作是一个文件，把它们的输入、输出等同于对磁盘文件的读和写。
键盘：标准输入文件　　屏幕：标准输出文件
其它设备：打印机、触摸屏、摄像头、音箱等
在 Linux 操作系统中，每一个外部设备都在/dev 目录下对应着一个设备文件，咱们在程序中要想操作设备，就必须对与其对应的/dev 下的设备文件进行操作。

标准 io 库函数对磁盘文件的读取特点



文件缓冲区是库函数申请的一段内存，由库函数对其进行操作，程序员没有必要知道存放在哪里，只需要知道对文件操作的时候的一些缓冲特点即可。
举例：放苹果到办公室的例子

1、行缓冲

标准 io 库函数，往标准输出（屏幕）输出东西的时候是行缓冲的

所谓的行缓冲就是缓冲区碰到换行符的时候才刷新缓冲区

刷新缓冲区的情况

1) 缓冲区里有换行符

```
例 1 :
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("hello world");
    while(1);
    return 0;
}
```

结果分析：printf 是标准 io 库函数，往标准输出输出东西，是行缓冲的，因为输出的东西没有'\n'
所以数据存放在缓冲区中，没有刷新缓冲区。故不能输出到屏幕上

```
例 2 :
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("hello world\n");
    while(1);
    return 0;
}
```

结果分析：hello world 打印到屏幕上，因为缓冲区中出现了'\n',刷新了缓冲区

2) 缓冲区满了，刷新缓冲区

```
例 3：
#include <stdio.h>
int main(int argc, char *argv[])
{
```

```
    while(1)
    {
        printf("hello world ");
    }
    return 0;
}
```

Printf 输出的内容虽然没有'\n'，但此程序也能将数据刷新到屏幕上，原因是缓冲满了，会自动刷新缓冲区。

3) 人为刷新缓冲区 fflush(stdout)

```
例 4：
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("hello world");
    fflush(stdout);//人为刷新缓冲区
    while(1);
    return 0;
}
```

结果分析：hello world 会打印到屏幕上，原因是 fflush(stdout),是人为的刷新缓冲区

4) 程序正常结束 会刷新缓冲区

```
例 5：
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("hello world");
    return 0;
}
```

2、全缓冲

标准 io 库函数 ，往普通文件读写数据的，是全缓冲的，
碰到换行符也不刷新缓冲区，即缓冲区满了，才刷新缓冲区

刷新缓冲区的情况

1.缓冲区满了，刷新缓冲区
2.人为刷新缓冲区 fflush(文件指针)
3.程序正常结束 会刷新缓冲区

3.无缓冲

在读写文件的时候通过系统调用 io （read write）,对文件进行读写数据
这个时候是无缓冲的，即写数据会立马进入文件，读数据会立马进入内存

写文件的流程

应用程序空间-→内核空间 -→驱动程序--→硬盘上

应用程序和内核程序运行在不同的空间里，目的是为了保护内核。

设置缓冲区的目的

通过缓冲可以减少进出内核的次数，以提高效率。

1.1.2 磁盘文件的分类

一个文件通常是磁盘上一段命名的存储区

计算机的存储在物理上是二进制的，

所以物理上所有的磁盘文件本质上都是一样的：以字节为单位进行顺序存储

从用户或者操作系统使用的角度（逻辑上）把文件分为：

文本文件：基于字符编码的文件

二进制文件：基于值编码的文件

文本文件

基于字符编码，常见编码有 ASCII、UNICODE 等

一般可以使用文本编辑器直接打开

例如：数 5678 的以 ASCII 存储形式为：

ASCII 码：00110101 00110110 00110111 00111000

歌词文件(lrc):文本文件

二进制码文件：

基于值编码,自己根据具体应用,指定某个值是什么意思

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

一般需要自己判断或使用特定软件分析数据格式

例如：数 5678 的存储形式为：

二进制码：00010110 00101110

音频文件(mp3):二进制文件

图片文件（bmp）文件，一个像素点由两个字节来描述*****######&&&&&

*代表红色的值

#代表绿色的值

&代表蓝色的值

二进制文件以位来表示一个意思。

文本文件、二进制文件对比：

译码：

文本文件编码基于字符定长，译码容易些；

二进制文件编码是变长的，译码难一些（不同的二进制文件格式，有不同的译码方式）。

空间利用率：

二进制文件用一个比特来代表一个意思(位操作)；

而文本文件任何一个意思至少是一个字符。

二进制文件，空间利用率高。

可读性：

文本文件用通用的记事本工具就几乎可以浏览所有文本文件

二进制文件需要一个具体的文件解码器，比如读 BMP 文件，必须用读图软件。

总结一下：

文件在硬盘上存储的时候，物理上都是用二进制来存储的。

咱们的标准 io 库函数，对文件操作的时候，不管文件的编码格式（字符编码、或二进制），而是按字节对文件进行读写，所以咱们管文件又叫流式文件，即把文件看成一个字节流。

1.2 文件指针

文件指针在程序中用来标示（代表）一个文件的，在打开文件的时候得到文件指针，
文件指针就用来代表咱们打开的文件。

咱们对文件进行读、写、关闭等操作的时候，对文件指针进行操作即可，即咱们将文件指针，传给读、写、关闭等函数，那些函数就知道要对哪个文件进行操作。

定义文件指针的一般形式为:

FILE * 指针变量标识符；

FILE 为大写，需要包含<stdio.h>

FILE 是系统使用 typedef 定义出来的有关文件信息的一种结构体类型，结构中含有文件名、文件状态和文件当前位置等信息

一般情况下，我们操作文件前必须定义一个文件指针**标示 我们将要操作的文件**

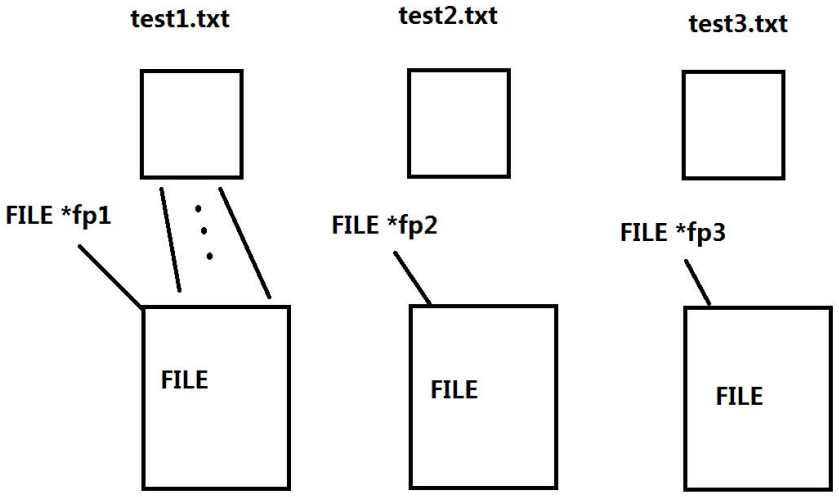
实际编程中使用库函数操作文件，无需关心 FILE 结构体的细节 ，只需要将文件指针传给 io 库函数，库函数再
通过 FILE 结构体里的信息对文件进行操作

FILE 在 stdio.h 文件中的文件类型声明：

```
typedef   struct
{   short level;              //缓冲区“满”或“空”的程度
    unsigned flags;           //文件状态标志
    char fd;                  //文件描述符
    unsigned charhold;        //如无缓冲区不读取字符
    short bsize;              //缓冲区的大小
    unsigned char *buffer;    //数据缓冲区的位置
    unsigned ar*curp;         //指针，当前的指向
    unsigned istemp;          //临时文件，指示器
    shorttoken;               //用于有效性检查
```

} FILE；

在缓冲文件系统中,每个被使用的文件都要在内存中开辟一块 FILE 类型的区域,存放与操作文件相关的信息



对文件操作的步骤：

1、对文件进行读写等操作之前要打开文件得到文件指针

2、可以通过文件指针对文件进行读写等操作

3、读写等操作完毕后，要关闭文件，关闭文件后，就不能再通过此文件指针操作文件了

c 语言中有三个特殊的文件指针无需定义，在程序中可以直接使用

stdin： 标准输入 默认为当前终端（键盘）

我们使用的 scanf、getchar 函数默认从此终端获得数据

stdout：标准输出 默认为当前终端（屏幕）

我们使用的 printf、puts 函数默认输出信息到此终端

stderr：标准错误输出设备文件 默认为当前终端（屏幕）

当我们程序出错使用:perror 函数时信息打印在此终端

1.3 打开文件 fopen

函数的定义：

FILE *fopen(const char *path, const char *mode);

函数说明：

fopen 函数的功能是打开一个已经存在的文件，并返回这个文件的文件指针（文件的标示）

或者创建一个文件，并打开此文件，然后返回文件的标示。

函数的参数：

参数 1:打开的文件的路径

1. 绝对路径,从根目录开始的路径名称

   “/home/edu/test/test.txt”

2. 相对路径

   “./test/test.txt”

   “../test/test.txt”

参数 2：文件打开的方式，即以什么样的方式（只读、只写、可读可写等等）打开文件

第二个参数的几种形式（打开文件的方式）

r w a +

模　式	功　能
r 或 rb	**以只读方式打开一个文本文件（不创建文件）**
w 或 wb	**以写方式打开文件（使文件长度截断为 0 字节，创建一个文件）**
a 或 ab	**以添加方式打开文件，即在末尾添加内容，当文件不存在时，创建文件用于写**
r+或 rb+	**以可读、可写的方式打开文件(不创建新文件)**
w+或 wb+	**以可读、可写的方式打开文件 （使文件长度为 0 字节，创建一个文件）**
a+或 ab+	**以添加方式打开文件，打开文件并在末尾更改文件（如果文件不存在，则创建文件）**

返回值：

成功：打开的文件对应的文件指针

失败：返回 NULL

以后调用 fopen 函数的时候，一定要判断一下，打开是否成功。

1.4 关闭文件 fclose

函数的头文件：

#include <stdio.h>

函数的定义:

int fclose(FILE *fp);

函数的说明：

关闭 fp 所代表的文件

注意一个文件只能关闭一次，不能多次关闭。关闭文件之后就不能再文件指针对文件进行读写等操作了。

返回值：

成功返回 0

失败返回非 0

注意：可以通过返回值，来判断关闭文件是否成功。

例 6：

```
#include<stdio.h>
int main()
{
```

```
    FILE *fp;
    int ret;
    fp=fopen("./test.txt","r+");
    if(fp==NULL)
    {
        perror("fopen");
        return 0;
    }
    printf("打开文件成功\n");
    ret=fclose(fp);
    if(ret==0)
        printf("关闭文件成功\n");
    else
        printf("关闭文件失败");
    return 0;
}
```

1.5 一次读写一个字符

函数定义：

int fgetc(FILE *stream);

函数说明：

fgetc 从 stream 所标示的文件中读取一个字节，将字节值返回

返回值：

以 t 的方式： 读到文件结尾返回 EOF

以 b 的方式：读到文件结尾，使用 feof(后面会讲)判断结尾

函数的定义：

int fputc(int c, FILE *stream)

函数的说明：

fputc 将 c 的值写到 stream 所代表的文件中。

返回值：

如果输出成功，则返回输出的字节值；

如果输出失败，则返回一个 EOF。

EOF 是在 stdio.h 文件中定义的符号常量，值为-1

注意：打开文件的时候，默认读写位置在文件的开始，如果以 a 的方式打开读写位置在文件的末尾

咱们向文件中读取字节或写入字节的时候，读写位置会往文件的末尾方向偏移，读写多少个字节，读写位置就往文件的末尾方向偏移多少个字节

例 7:

```
#include <stdio.h>
int main(void)
{
	FILE *fp;
	char ch;
	fp=fopen("test.txt","r+");
	if(fp==NULL)
	{
		printf("Cannot open the file\n");
		return 0;
	}
	while(   (ch = fgetc(fp))!=EOF   )
	{
		fputc(ch,stdout);
	}
	fclose(fp);
	return 0;
}
```

1.6 一次读写一个字符串

函数的定义：

char *fgets(char *s, int size, FILE *stream);

函数的说明：

从 stream 所指的文件中读取字符，在读取的时候碰到换行符或者是碰到文件的末尾停止读取，或者是读取了 size-1 个字节停止读取，在读取的内容后面会加一个\0,作为字符串的结尾

返回值：

成功返回目的数组的首地址，即 s

失败返回 NULL

函数的定义：

int fputs(const char *s, FILE *stream);

函数的说明：

将 s 指向的字符串，写到 stream 所代表的文件中

返回值：

成功返回写入的字节数

失败返回 -1

例 9：

```
#include <stdio.h>
int main(void)
{
    FILE *fp_read,*fp_write;
    char string1[100];
    if((fp_read=fopen("src.txt","r+"))==NULL)
    {
        printf("Cannot open the file\n");
        return 0;
    }
    if((fp_write=fopen("dest.txt","w+"))==NULL)
    {
        printf("Cannot open the file\n");
        return 0;
    }
    fgets(string1, 100, fp_read);
    printf("%s\n",string1);
    fputs(string1,fp_write);
    fclose(fp_read);
    fclose(fp_write);
    return 0;
}
```

1.7 读文件 fread

函数的定义：

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

函数的说明：

fread 函数从 stream 所标示的文件中读取数据，一块是 size 个字节，共 nmemb 块，存放到 ptr 指向的内存里

返回值：

实际读到的块数。

例 1：

int num;

num=fread(str,100,3,fp);

从 fp 所代表的文件中读取内容存放到 str 指向的内存中，读取的字节数为 ，每块 100 个字节，3 块。

返回值 num，

如果读到 300 个字节返回值 num 为 3

如果读到了大于等于 200 个字节小于 300 个字节 返回值为 2

读到的字节数，大于等于 100 个字节小于 200 个字节 返回 1

不到 100 个字节返回 0

1.8 写文件 fwrite

函数的定义：

size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);

函数的说明：

fwrite 函数将 ptr 指向的内存里的数据，向 stream 所标示的文件中写入数据，一块是 size 个字节，共 nmemb 块。

返回值：

实际写入的块数

例 10：

```
#include <stdio.h>
struct stu
{
    char name[10];
    int num;
    int age;
}boya[10],boyb[2];
int main()
{
    FILE *fp;
    int i;
    if((fp=fopen("test.txt","wb+"))==NULL)
    {
        printf("Cannot open file!");
        return 0;
    }
    printf("input data\n");
    printf("name、num、age\n");
    for(i=0;i<2;i++)
        scanf("%s %d %d",boya[i].name,&boya[i].num,&boya[i].age);

    fwrite(boya,sizeof(struct stu),2,fp);      //将学生信息写入文件中
    rewind(fp);   //文件指针经过写操作已经到了最后，需要复位
    fread(boyb,sizeof(struct stu),2,fp);      //将文件中的数据读入到内存中

    for(i=0;i<2;i++)
        printf("%s %d %d\n",boyb[i].name,boyb[i].num,boyb[i].age);
    fclose(fp);
```

```
    return 0;
}
```

注意：

fwrite 函数是将内存中的数据原样输出到文件中。

fread 函数是将文件中的数据原样读取到内存里。

1.9 格式化读写文件函数

函数调用:

fprintf (文件指针，格式字符串，输出表列）；

fscanf (文件指针，格式字符串，输入表列）；

函数功能:

从磁盘文件中读入或输出字符

fprintf 和 printf 函数类似：

printf 是将数据输出到屏幕上（标准输出），

fprintf 函数是将数据输出到文件指针所指定的文件中。

fscanf 和 scanf 函数类似：

scanf 是从键盘（标准输入）获取输入，

fscanf 是从文件指针所标示的文件中获取输入。

例 11：

```
#include <stdio.h>
int main(void)
{
    FILE *fp;
    char ch1='a',ch2;
    int num1=50,num2;
    char string1[20]="hello",string2[20];
    float score1=85.5,score2;

    if((fp=fopen("test.txt","wb+"))==NULL)
    {
        printf("Cannot open the file\n");
        return 0;
    }
    fprintf(fp,"%c %d %s %f\n",ch1,num1,string1,score1);
```

```
    rewind(fp);
    fscanf(fp,"%c %d %s %f\n",&ch2,&num2,&string2,&score2);
    printf("%c %d %s %f\n",ch2,num2,string2,score2);
    fclose(fp);
    return 0;
}
```

1.10 随机读写

前面介绍的对文件的读写方式都是顺序读写，即读写文件只能从头开始，顺序读写各个数据;
但在实际问题中常要求只读写文件中某一指定的部分，例如：读取文件第 200--300 个字节

为了解决这个问题可以移动文件内部的位置指针到需要读写的位置，再进行读写，这种读写称为随机读写
实现随机读写的关键是要按要求移动位置指针，这称为文件的定位.

完成文件定位的函数有：

rewind、fseek 函数

1、 rewind 复位读写位置

rewind 函数

void rewind(文件指针);

函数功能:

把文件内部的位置指针移到文件首

调用形式:

rewind(文件指针);

例 12：

```
fwrite(pa,sizeof(struct stu),2,fp ) ;
rewind(fp);
fread( pb,sizeof(struct stu),2,fp);
```

2、 ftell 测文件读写位置距文件开始有多少个字节

定义函数:

long ftell(文件指针);

函数功能:

取得文件流目前的读写位置.

返回值:

返回当前读写位置(距离文件起始的字节数)，出错时返回-1.

- 例如：

```
int length;
 length = ftell(fp);
```

3、 fseek 定位位置指针（读写位置）

fseek 函数（一般用于二进制文件即打开文件的方式需要带 b）

定义函数:

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

//int fseek(文件类型指针，位移量，起始点);

函数功能：

移动文件流的读写位置.

参数：

whence 起始位置

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、后移动的字节数，正数往文件末尾方向偏移，负数往文件开头方向偏移。

例 13：

```
fseek(fp,50,SEEK_SET)
fseek(fp,-50,SEEK_END);
fseek(fp,0,SEEK_END);
```