

# 文件的随机读写

- 如何实现文件的随机读写呢？
- 文件从头到尾按字节从0开始顺序编址，用以表示数据的存储位置
- 文件位置指针(**Position pointer, 文件位置标记**)
  - \* 指示打开文件的当前读写位置
  - \* 对文件每读写一次，文件指针自动指向下一个读写位置，可方便地进行顺序读写
  - \* 利用文件定位函数，还可实现随机读写

# 文件的随机读写

## ■ 文件定位: 设定文件位置指针

- \* 使文件位置指针重新指向文件的开始位置

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

- \* 返回当前文件位置指针(相对于文件起始位置的字节偏移量)

```
long ftell(FILE *fp);
```

- \* 改变文件位置指针, 实现随机读写

```
int fseek(FILE *fp, long offset, int fromwhere);
```

# 文件的随机读写

## ■ fseek

- \* 将文件位置指针从**fromwhere**开始移动**offset**个字节, 指示下一个要读取的数据的位置

**fp:** 指向**fopen()** 打开的文件的指针

**offset:** 位置指针的偏移量(字节数), **long**型, 加**1**或**L**

**fromwhere:** 起始位置

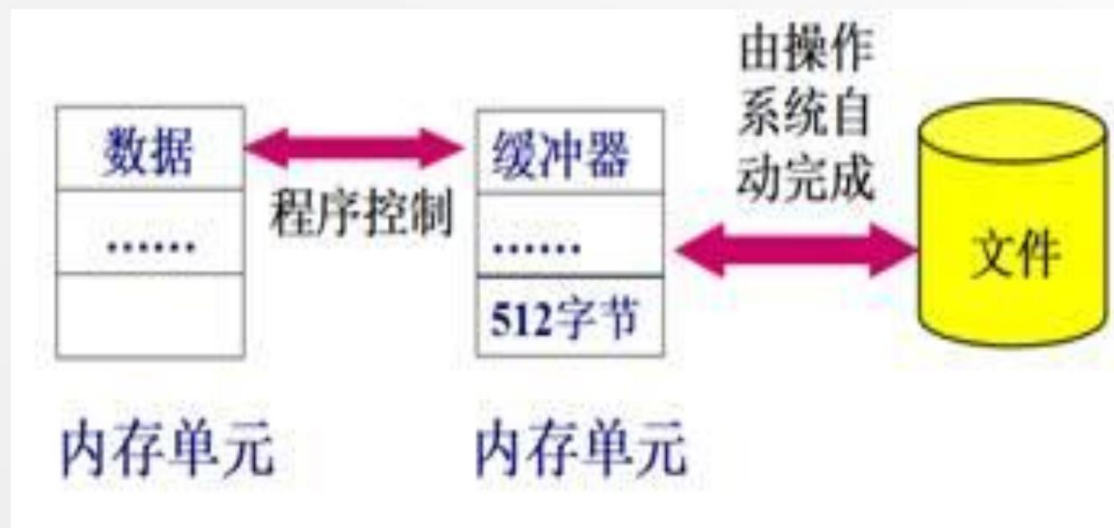
**SEEK\_SET**或**0**----文件开始

**SEEK\_CUR**或**1**----当前位置

**SEEK\_END**或**2**----文件末尾

# 文件缓冲

- 从磁盘驱动器传出或传入信息，相对较慢
- 提高读写效率的诀窍
  - \* 缓冲(buffering)



- 向磁盘输出数据: 数据—> 缓冲区, 装满缓冲区后—> 磁盘文件
- 从磁盘读入数据: 先一次性从磁盘文件将一批数据输入到缓冲区, 然后再从缓冲区逐个读入数据到变量

# 文件缓冲

## ■ 输出缓冲

- \* 写入文件的数据, 实际是存储在内存的缓冲区内
- \* 当缓冲区满或关闭文件时, 缓冲区自动“清洗”(写入输出设备)
- \* 主动清洗输出流, 用 `fflush(fp)` ;

## ■ 输入缓冲

- \* 来自输入设备的数据存在输入缓冲区内
- \* 从缓冲区读数据代替了从设备本身读数据

# 文件缓冲

- 文件缓冲提高效率的原因所在
  - \* 缓冲发生在屏幕的后台, 自动完成, 如getchar()
  - \* 从缓冲区读数据或向缓冲区写数据, 几乎不花时间
  - \* 仅花时间将缓冲区内容传递给磁盘文件, 或反之
  - \* 一次性的大块移动比频繁的字符移动快得多

# 缓冲型和非缓冲型文件系统

## ■ 缓冲型文件系统

- \* 系统自动在内存中为每个正在使用的文件开辟一个缓冲区，读写文件时，数据先送缓冲区再传给C程序或外存
- \* 利用文件指针标识文件
- \* 缓冲型文件系统下的文件操作，也称高级文件操作
- \* 高级文件操作函数是ANSI C定义的文件操作函数，具有跨平台和可移植的能力

## ■ 非缓冲文件系统

- \* 不自动设置文件缓冲区，缓冲区需由程序员自己设定
- \* 没有文件指针，使用称为文件号的整数来标识文件

# 两种方式的区别

fopen族的函数	open族的函数
将很多功能从不同OS的范畴转移到了标准语言库的范畴, 实现了以独立于实现的方式来执行文件I/O	功能一般由OS直接提供, 在不同的OS上有细微差别
较适合处理文本文件, 或结构单一的文件, 会为了处理方便而改变一些内容	通常能直接反映文件的真实情况, 因为它的操作都不假定文件的任何结构
功能更强大, 但效率略逊	