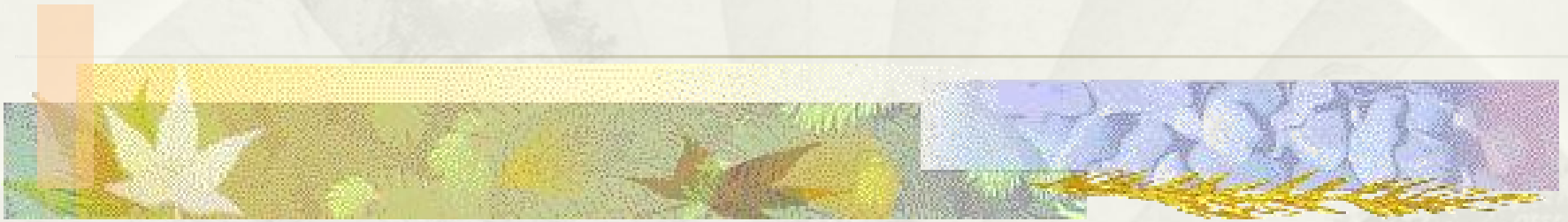
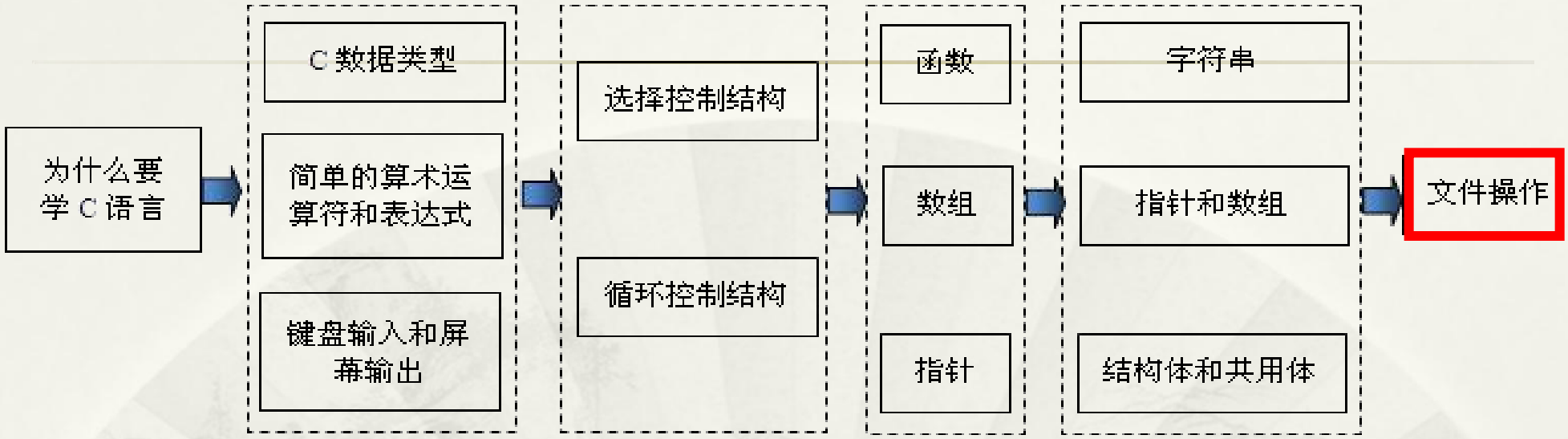
哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

规格严格 功夫到家

第13章 文件操作



哈尔滨工业大学（深圳）
计算机科学与技术学院
刘洋
Liu.yang@hit.edu.cn



简单的数据结构 → 复杂的数据结构

自底向上的程序设计方法 → 自顶向下的程序设计方法

第13章 文件操作 学习内容

- C语言中的流
- 标准输入输出及其重定向
- 文件的分类
- 文件的打开和关闭
- 文件的读写

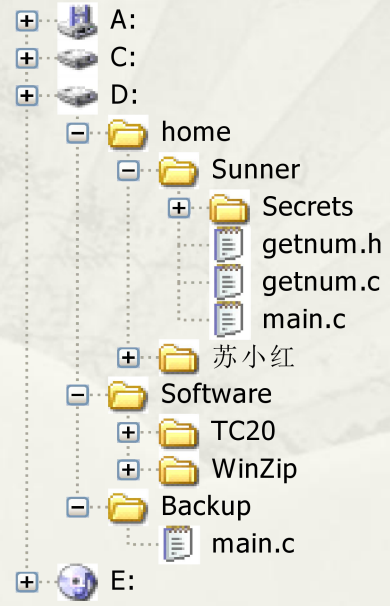
何谓文件（*Files*）？

- 存储在外部介质上具有名字的一组相关数据的集合
  - **数据**一般以文件的形式为用户及应用程序使用
  - 文件1、泛指磁盘**文件**；2、 OS对于例如终端显示器或打印机等**设备**，以文件形式来管理的。
- 为什么要使用文件？

1、程序与数据分离

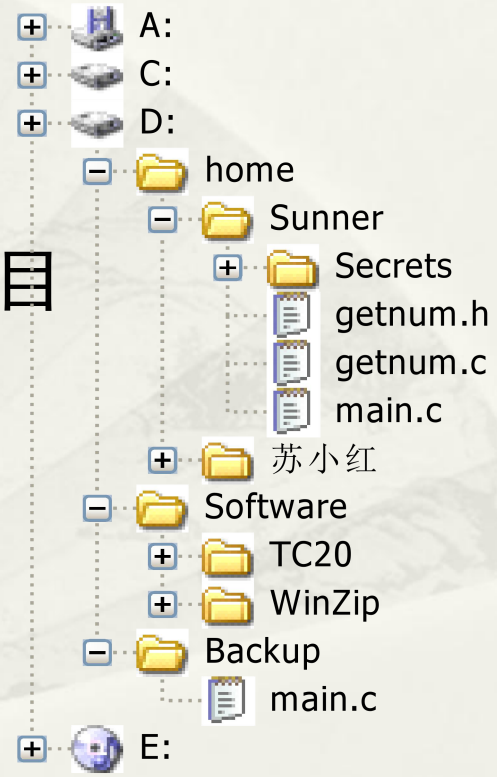
2、数据共享

3、长期保存数据



文件存储的方式

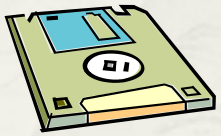
- 建立若干目录(文件夹,directory\folder)
  - 在目录里保存文件
  - 同一级目录里保存的文件不能同名
- 对使用者而言，只要知道文件的路径(path, 全目录)和文件名，就能使用该文件
  - D:\home\Sunner\main.c
  - Windows用反斜杠\分割路径
  - UNIX用斜杠/



将文件存储在外存

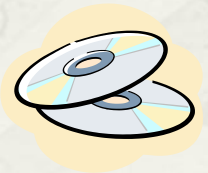
- 外存
  - 内存容量小，容易健忘，掉电即失
  - 外存容量大、断电后数据不丢失，可重复使用，永久保存，不健忘

磁盘（Magnetic disks）



  - 磁盘表面涂有磁性物质
  - 磁性单元的N-S极的两种指向表示0-1

光盘（CD、DVD）



  - 光盘表面有一层特殊介质
  - 介质的高低不平的交替表示0-1

U盘（Flash Memory）

  - 闪存，可用电擦除的ROM
  - 一种电化学存储介质，电流的通断表示0-1

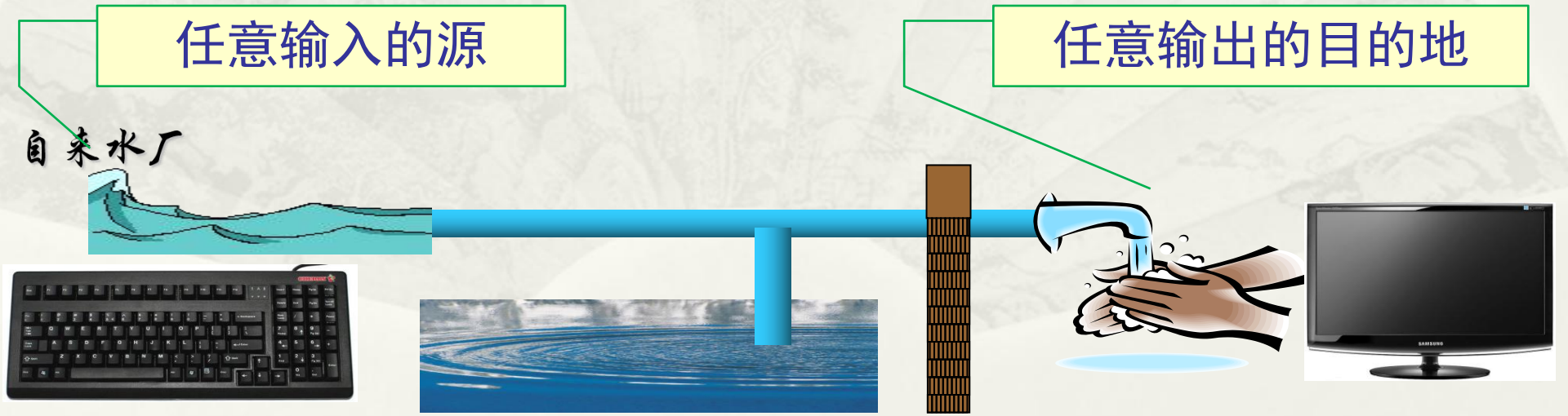
文件的分类

- 1. 按文件的逻辑结构
- 记录文件
  - 由具有一定结构的记录组成（定长和不定长）
- 流式文件 stream
  - 由一个个字符（字节）数据顺序组成
  - 数据流，字节流

流（Stream）——计算机的基础设施 infrastructure

正如日常生活中：水、电、煤气、服务等基础设施，流成为…

- 通过一个流（通常和键盘相关）获得全部的输入
- 通过另一个流（通常和屏幕相关）写出全部的输出
- 较大规模的程序
  - 可能需要额外的流，如磁盘文件、网络端口、打印机等



C语言中的标准流

- stdio.h提供了三种标准流
  - 系统定义三个标准文件，分别对应于终端设备：stdin、stdout、stderr。它们共同构成了计算机的基础设施，是备用的，不能声明、打开和关闭

文件指针	标准流	默认的含义
stdin	标准输入	键盘
stdout	标准输出	终端显示器屏幕
stderr	标准错误输出	终端显示器屏幕

C语言中的标准流

文件指针	标准流	默认的含义
stdin	标准输入	键盘
stdout	标准输出	终端显示器屏幕
stderr	标准错误输出	终端显示器屏幕

- **scanf(), getchar(), gets()**等通过**stdin**获得输入
- **printf(), putchar(), puts()**等用**stdout**进行输出

Tips: **stdout**和**stderr**在默认情况下都指向屏幕，有何区别呢？

输出到**stdout**的内容先保存到缓冲区

而输出到**stderr**的内容则直接输出到屏幕

输入/输出重定向 redirection

- 某些操作系统允许标准输入/输出重定向文件
  - DOS和UNIX允许程序从文件获得输入或向文件写数据
    - 这种重定向，程序本身是感觉不到的
- 输入重定向（Input Redirection）
  - `D:\>demo < infile.txt`
  - 从终端（键盘）输入数据改成从文件中读入数据
- 输出重定向（Output Redirection）
  - `D:\>demo > outfile.txt`
  - 从终端（显示器）输出数据重定向到文件

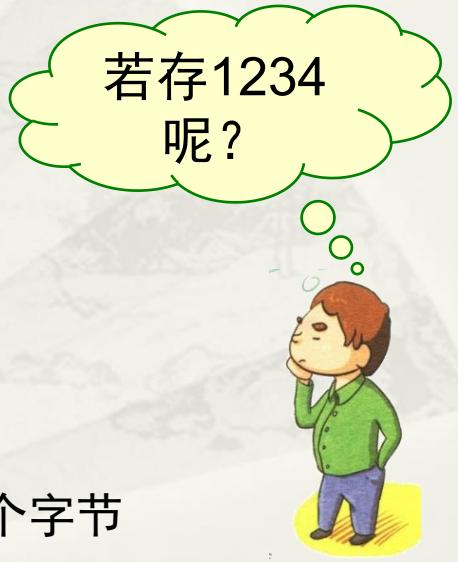
文件的分类

- 2. 文件按数据的组织形式可以分为：
  - 文本文件（Text file）
  - 二进制文件（Binary file）
    - C程序的源代码
      - 文本文件
    - 可执行的C程序
      - 二进制文件
  - 文本文件和二进制文件有什么区别呢？

文件的分类

- 文本文件
  - 用字节表示字符的字符序列，存储每个字符的ASCII码
    - 如：整数123在文本文件中占3个字节，分别存放这3个字符的ASCII码

字符：	'1'	'2'	'3'
十进制的ASCII值：	49	50	51
二进制的ASCII值：	00110001	00110010	00110011

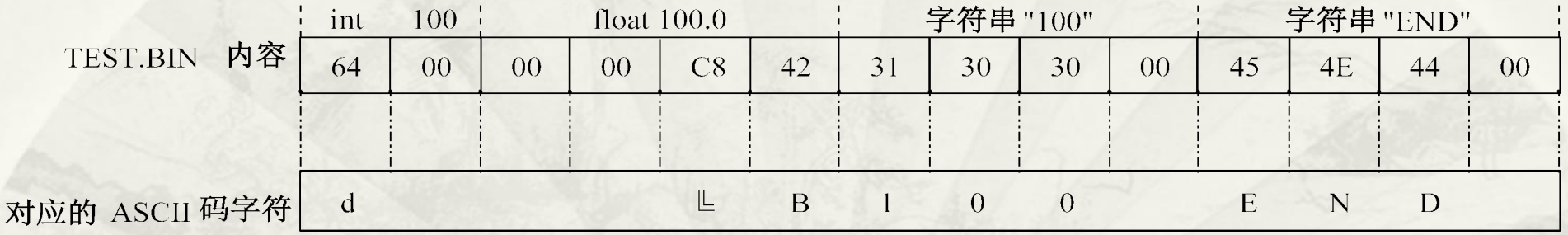


- 二进制文件
  - 如短整型数123，在内存占2个字节，在二进制文件中也占2个字节
  - 节省空间

00000000	01111011

13.1二进制文件和文本文件

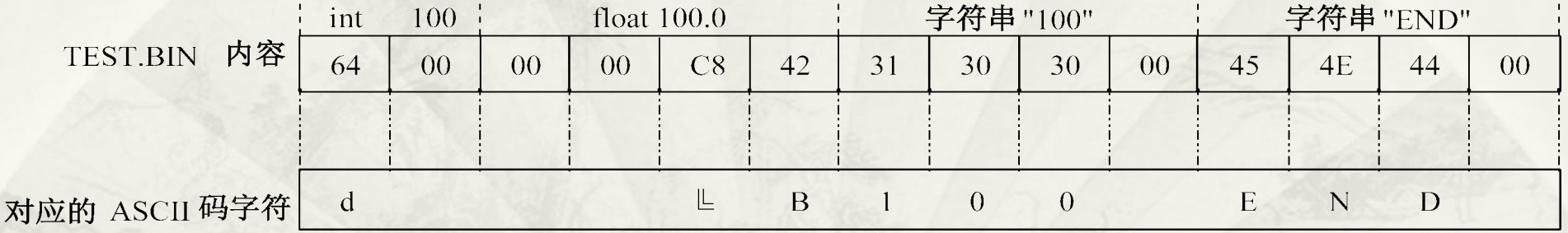
- 二进制文件与文本文件的区别：
  - 按照数据在内存中的存储形式（二进制）存储到文件
  - 二进制文件存储的字节不一定表示字符，无需ASCII码表进行字符变换，读写速度快



- Tips: 必须按照数据存入的类型和格式读出才能恢复本来面貌
  - 先按int类型读，为0x0064，是整数100
  - 而按float读，将读出0x00000064，对应的float值为1.4012985e-43，近似为0，面目全非

13.1二进制文件和文本文件

- 二进制文件
  - 字节不一定表示字符，无需ASCII码表与字符变换读写速度快
  - 按照数据在内存中的存储形式（二进制）存储到文件



- 缺点
  - 不易阅读，妨碍调试过程
  - 尤其注意可移植性问题：

    二进制文件与不同类型的计算机及其存储数据方式相关。
    - 存储int型2字节,4字节？先存高位字节,低位字节？与机器字长、对齐方式alignment以及big-endian / little-endian等相关。

13.1二进制文件和文本文件

- 总之，二进制文件保存数据，不仅要按照存入时的类型，还要按存入时的格式读出，才能恢复其本来面貌。

- 公开的标准文件格式
  - 如bmp、tif、gif、jpg和mp3等类型的文件，有大量软件生成和使用这些类型的文件
- 不公开或加密的文件格式
  - 如Microsoft Word的doc格式就不公开，所以至今没有Word以外的其他软件能完美地读出doc文件

13.1二进制文件和文本文件

- 实际上，文件就是一种字节序列，无论何种形式。
- 对于程序员来说，区分文本文件和二进制文件：
- 文本文件——按行划分
  - 所以必须用特殊的字符标记行的结尾
  - 某些OS还可能用特殊的字符标记文件的末尾
    - 例如，DOS将Ctrl+Z设定为文件的结束符
- 二进制文件——不是按行划分的
  - 可合法地包含任何字符，故不可能留出文件结束符

文件与流的关系

- 程序**通过文件打开操作将流与设备联系起来**，文件打开后，可在程序和文件之间交换数据
  - 由程序在磁盘上建立文件（建立文件）
  - 文件打开后，通过写操作将数据存入该文件（写入数据）
  - 由程序打开磁盘上的某个已有文件，通过读操作将文件中的数据读入内存供程序使用（读取数据）
- 程序通过文件关闭操作断开流与文件的联系

文件指针（File Pointer）

- C程序中流的打开和关闭是通过文件指针实现的
- 文件指针的类型为**FILE ***
- **FILE * fp ;**
  - 定义了**FILE**型指针变量*fp，标识一个特定的磁盘文件
  - Tips：与文件相关联的每个流都有一个**FILE**类型的控制结构，定义有关文件操作的信息，用户绝对不应修改

文件指针（File Pointer）

- 文件指针类型在stdio.h中定义

```
typedef struct                      /*在stdio.h文件中定义*/
{
    short level;                    /*fill/empty level of buffer*/
    unsigned flags;                 /*File status flags*/
    char fd;                        /*File descriptor*/
    unsigned char hold;             /*Ungetc char if no buffer*/
    short bsize;                    /*Buffer size*/
    unsigned char *buffer;          /*Data transfer buffer*/
    unsigned char *curp;            /*Current active pointer*/
    unsigned istemp;                /*Temporary file indicator*/
    short token;                    /*Used for validity checking*/
}FILE;                              /*This is the FILE object*/
```

13.2 文件的打开

- 格式：`FILE *fopen( const char *filename, const char *mode);`
  - `FILE *fp;//返回打开/生成文件的文件指针`
  - `fp = fopen("test.txt", "r");`
- **filename**是文件名
  - 包含路径。如果不含路径，表示打开当前目录下的文件
  - `fp = fopen("D:\newproject\test.txt", "r");`
  - 编译器会将'\'看成转义字符，例如：\n和\t，为此“\\”
  - `fp = fopen("D:\\newproject\\test.txt", "r");`

13.2文件的打开

- 格式：`FILE *fopen(const char *filename, const char *mode);`
  - `FILE *fp;`
  - `fp = fopen("test.txt", "r");`
- **返回值为指向此文件的指针** *fp
  - 如果打开失败（文件损坏或不存在），返回值为**NULL**
- Tips: **文件打开后一定要检查是否打开成功**
  - 通过测试**fopen**的返回值，判断文件打开是否成功：

```
if (fp == NULL)
{
    printf("Failure to open test.txt!\n");
    exit(0);
}
```

13.2文件的打开

- 格式：`FILE *fopen(const char *filename, const char *mode);`
  - `FILE *fp;`
  - `fp = fopen("test.txt", "r");`
- 返回值为指向此文件的指针
  - 如果打开失败（文件损坏或不存在），返回值为**NULL**
- Tips: 为什么要返回文件指针（句柄,handle）呢?
  - 以后关闭文件时要使用
  - 因FILE结构较大，传回这个结构的首地址的效率比传回整个结构效率高

13.2文件的关闭

- 格式：`int fclose(FILE *fp);`
  - 把遗留在缓冲区中的数据写入文件，实施操作系统级的关闭操作
  - 同时，释放与流联系的文件控制块FCB ，以便以后重复使用
- `fclose`函数的返回值
  - **若成功执行了关闭操作，返回值为0**
  - 否则返回为**非零值**，表示关闭时有错误

  例如：驱动器中无盘或盘空间不够时文件操作失败，文件关闭失败会导致数据丢失、文件破坏，甚至程序出现随机错误

13.2文件的关闭

- 注意tips：文件用完一定要关闭。
  - 否则，可能引起数据丢失
  - 甚至影响其他文件的打开

多数情况下，系统限制同时打开状态的文件总数。

因此，打开文件前先关闭无用文件是必要的。

13.2文件的打开和关闭

- FILE *fopen(const char *filename,
  const char *mode);
  - FILE *fp;
  - fp = fopen("test.txt", "r");

- filename:文件名
- 返回值:指向此文件的指针
  - **mode**:文件打开方式

13.2文件的打开和关闭

文件打开方式（mode）

r	只读	必须是已存在的文件
w	只写	**无论该文件是否存在**，都新建一个文件
a	追加	向文本文件尾添加数据，**该文件必须已经存在**
r+	读写	打开一个**已存在的文件**，用于读写
w+	读写	**建立一个新文件**，可读可写
a+	读写	向文件尾追加数据，也可读

对应文本文件

rb
wb
ab
rb+
wb+
ab+

以二进制方式打开文件
原汁原味地体现文件内容

13.2文件的打开和关闭

文件打开方式（mode）

r	只读	必须是已存在的文件
w	只写	不论该文件是否存在，都新建一个文件
a	追加	向文本文件尾添加数据，该文件必须存在
r+	读写	打开一个已存在的文件，用于读写
w+	读写	建立一个新文件，可读可写
a+	读写	向文件尾追加数据，也可读

rb
wb
ab
rb+
wb+
ab+

若文件不存在，w新建一个文件，
若文件存在，w会将原文件内容覆盖

用a打开文件，要求该文件必须存在，
保留原文件内容，在文件末尾添加

w和a的区别？

13.2文件的打开和关闭

文件打开方式（mode）

r	只读	必须是已存在的文件
w	只写	不论该文件是否存在，都新建一个文件
a	追加	向文本文件尾添加数据，该文件必须存在
r+	读写	打开一个已存在的文件，用于读写
w+	读写	建立一个新文件，可读可写
a+	读写	向文件尾追加数据，也可读
rb		
wb		
ab		
rb+		
wb+		
ab+		

若文件不存在：r+打开会失败，
w+则会新建一个文件

若文件存在：r+不清空文件，
w+则清空文件，因此
当文件存在时要慎用w+

r+和w+的区别？

13.2文件的打开和关闭

文件打开方式（mode）

r	只读	必须是已存在的文件
w	只写	不论该文件是否存在，都新建一个文件
a	追加	向文本文件尾添加数据，该文件必须存在
r+	读写	打开一个已存在的文件，用于读写
w+	读写	建立一个新文件，可读可写
a+	读写	
rb		
wb		
ab		
rb+		
wb+		
ab+		

文本文件

字符读写：fgetc(), fputc()字符

fgets(), fputs()字符串

格式化读写：fscanf(), fprintf()

二进制文件

数据块读写：fread(), fwrite()

13.3 按字符读写文件

- 按字符读写
- **int fgetc(FILE *fp);**
  - 每从fp读出一个字符，位置指针自动指向下一个字符
  - 若读成功，则返回该字符（返回值为int，非char）
  - 若读到文件尾，则返回EOF

  EOF在stdio.h中定义为-1，读文件时自动添加到缓冲区

  int fputc(int c, FILE *fp);

  *向fp输出字符c

  *若写入错误，则返回EOF，否则返回c (int非char)

13.3按字符读写文件

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   int  main()
4   {
5       FILE *fp;
6       char ch;
7       if ((fp = fopen("demo.txt","w")) == NULL) /* 判断文件是否成功打开 */
8       {
9           printf("Failure to open demo.txt!\n");
10          exit(0);
11      }
12      ch = getchar();
13      while (ch != '\n')
14      {
15          fputc(ch, fp);
16          ch = getchar();
17      }
18      fclose(fp);
19      return 0;
20  }
```

文件打开

```
fp = fopen("demo.txt", "w");
if (fp == NULL)
{
    printf("Failure to open demo.txt!\n");
    exit(0);
}
```

文件读写操作

文件关闭

【例13.1】从键盘键入一串字符，转存到磁盘文件上demo.txt

13.3按字符读写文件

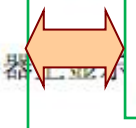
【例13.2】

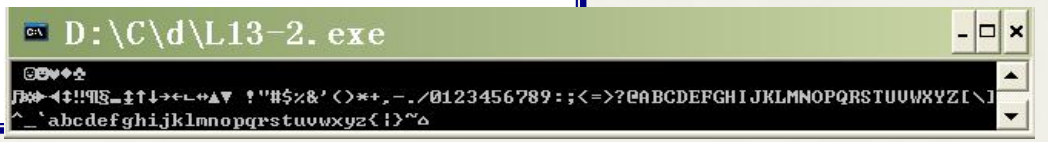
将0~127之间的ASCII字符写入demo.bin二进制文件中，并读取该文件内容并显示（用写字板和vc打开）。

```c
int  main()
{
    FILE *fp;
    char ch;
    int i;
    if ((fp = fopen("demo.bin","wb")) == NULL)
    {
        printf("Failure to open demo.bin!\n");
        exit(0);
    }
    for (i=0; i<128; i++)
    {
        fputc(i, fp);     /* 将 ASCII 码值在 0-127 之间的所有字符写入文件 */
    }
    fclose(fp);
    if ((fp = fopen("demo.bin","rb")) == NULL) /* 以二进制读方式打开文件 */
    {
        printf("Failure to open demo.bin!\n");
        exit(0);
    }
    while ((ch = fgetc(fp)) != EOF)     /* 读取文件内容，直到文件尾 */
    {
        putchar(ch);          /* 在显示器上显示 */
    }
    fclose(fp);
    return 0;
}
```

函数feof()检查是否到达文件尾，当文件位置指针指向文件尾时，返回非0值，否则返回0值

```c
ch = fgetc(fp);
while (!feof(fp))
{
    putchar(ch);
    ch = fgetc(fp);
}
```





】7之CII字符写入demo.bin二进制文件中，并读取内容显示（用写字板和vc打开）。

13.3按字符串读写文件

- 字符串（行）读

```
char *fgets(char *s, int n, FILE *fp);
```

  - 从fp所指的文件中读取字符串，并在字符串末尾添加'\0'，然后存入s中，最多读n-1个字符
  - 当读到回车换行符、文件末尾，或读满n-1个字符时，函数返回该字符串的首地址
  - 特例：`fgets(buf, sizeof(buf), stdin);`
    - 从标准输入流（键盘）中读数据——替换gets(buf)，对于防止缓冲区溢出攻击防御性程序设计defensive programming，具有重要意义。

13.3按字符串读写文件

- 字符串（行）写

```
int fputs(const char *s, FILE *fp);
```

* 将字符串s写入fp所指的文件中
* 需要注意：

  与puts()不同的是它不会自己写入换行符，除非字符串本身含有换行符；

  若出现写入错误，则返回EOF，否则返回一个非负数。

实例【13.4】

13.4按格式读写文件

- **fprintf()**的最普遍的应用就是
  - 向标准错误流写出错信息

```
fprintf(stderr,"Error: file can't be opened\n");
```

    - 即使用户重定向**stdout**，向**stderr**写入的信息也保证会出现在屏幕上
  - 输出调试信息，报告程序运行到了哪里，那里的状态是什么

```
int Add(int para1, int para2)
{
 int a, b;
 fprintf(stderr,"DEBUG: Call foo() with %d and %d\n",para1,para2);
 a = para1;
 b = para2;
 return a + b;
}
```

13.5按数据块读写文件

- 按数据块读

```
num = fread(buffer, size, count, fp);
```

  - 从fp所指文件中读取数据块存到buffer指向的内存
  - buffer是存储数据块的内存首地址，如数组的地址
  - size是每个数据块大小(元素的大小，字节数)
  - count是要读入的数据块个数 ，如数组元素的个数
  - 返回值num：实际读入的数据块个数
    - 应等于count，除非达到了文件末尾，或出现了错误

```
num = fread(a, sizeof(char), n, fp); //n个块，每个块占1字节
num = fread(a, sizeof(a[0]), sizeof(a)/sizeof(a[0]), fp);
num = fread(a, sizeof(a), 1, fp);     //1个块
```

13.5按数据块读写文件

- 按数据块写

```
num = fwrite(buffer, size, count, fp);
```

  - 将buffer指向的内存中的数据块写入fp所指的文件
  - buffer是数据块的首地址，如数组的地址
  - size是每个数据块大小（元素的大小，字节数）
  - count是要写入的数据块个数 ，如数组元素的个数
  - 返回值num：实际写入的数据块个数
    - 应等于count，除非出现写入错误

例如，若要写入整个数组：

```
num = fwrite(a, sizeof(a[0]), sizeof(a)/sizeof(a[0]), fp);
```

【例13.7】在前几个实例基础上，计算每个学生的4门课程的平均分，将学生的各科成绩及平均分输出到文件student.txt中，然后再从文件中读出数据并显示到屏幕上

```
71  void WritetoFile(STUDENT stu[], int n)
72  {
73      FILE *fp;
74      if ((fp = fopen("student.txt","w")) == NULL)
75      {
76          printf("Failure to open student.txt!\n");
77          exit(0);
78      }
79      fwrite(stu, sizeof(STUDENT), n, fp);
80      fclose(fp);
81  }
```

允许程序在单步中读和写比较大的数据块

虽然可以用于文本流，但主要还是用于二进制流

【例13.7】在前几个实例基础上，计算每个学生的4门课程的平均分，将学生的各科成绩及平均分输出到文件student.txt中，然后再从文件中读出数据并显示到屏幕上

```
83    int ReadfromFile(STUDENT stu[])
84    {
85        FILE *fp;
86        int i;
87        if ((fp = fopen("student.txt","r")) == NULL)
88        {
89            printf("Failure to open student.txt!\n");
90            exit(0);
91        }
92        for (i=0; !feof(fp); i++)
93        {
94            fread(&stu[i], sizeof(STUDENT), 1, fp);
95        }
96        fclose(fp);
97        printf("Total students is %d.\n", i-1);
98        return i-1;
99    }
```

13.6 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

13.6 文件的随机读写

文件打开方式（mode）

r	只读	必须是已存在的文件
w	只写	不论该文件是否存在，都新建一个文件
a	追加	向文本文件尾添加数据，该文件必须存在
r+	读写	打开一个已存在的文件，用于读写
w+	读写	建立一个新文件，可读可写
a+	读写	向文件尾追加数据，也可读

rb
wb
ab
rb+
wb+
ab+

不建议以读写方式打开文件！！！
每次读写都会改变文件位置指针，
很容易写乱，破坏原来文件的内容
需调用文件定位函数才能在读写之间转换
因读写其实公用一个缓冲区

文件定位

- 设定文件位置指针的函数

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

- 使文件位置指针重新指向文件的开始位置

```
long ftell(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）

文件定位

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

- 改变文件位置指针，实现随机读写
  - 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

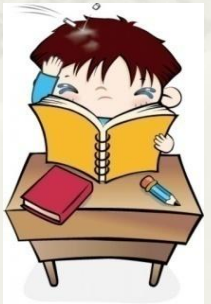
offset: 位置指针的偏移量（字节数），**long**型，加**1**或**L**

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）
- 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
  - 缓冲型文件系统中的文件操作，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	

小结

- 文件的分类
- 文件操作函数
  - 文件的打开和关闭
  - 文件的顺序读写
  - 文件的随机读写

小结

- 有没有图形的标准函数呢？
  - 没有
  - 编译器可能带有图形库
  - 使用第3方提供的图形库
  - 编写自己的图形库

C语言的核心学习到此结束

- 常用的32个关键字和围绕它们的语法构成了C语言的核心
  - 26个字母和围绕它们的构词法、语法构成了英语的核心
- 能背下英语的所有单词和语法，就能写出莎士比亚一样的诗句吗？
  - 仅掌握语言的核心是远远不够的
  - 本课程对C语言核心的讲述也并非面面俱到，而是核心的核心，以及有代表性的外延
  - 写出的漂亮程序不是听课听出来的，也不是看书看出来的，而是读别人的程序读出来的，更是自己动手练出来的


Thank you for
your attention !
SuXiaoHong


Q&A