# step further

文件 **专题**

**起步：**

认知与体验（硬件、软件、程序与C语言）

**进阶：**

判断与推理（流程控制方法、语句）

抽象与封装（模块设计方法、函数）

表达与转换（基本操作、数据类型）

**提高：**

构造与访问（数组、指针、**结构**）

归纳与推广（程序设计的本质）

- 概述
- 文件类型指针
- 文件的打开
- 文件的读写
- 文件的定位
- 文件的关闭
- 注意事项

# 文件概述

- 程序运行结果有时需要永久性地保存起来，以供其他程序或本程序下一次运行时使用。程序运行所需要的数据也常常要从其他程序或本程序上一次运行所保存的数据中获得。

- 用于永久性保存数据的设备称为外部存储器（简称：外存），如：磁盘、磁带、光盘等。
- 在外存中保存数据的方式通常有两种：文件和数据库。本课程只介绍文件方式。

- 计算机中的文件是一种数据集合，每个文件由若干个数据项序列组成，操作系统将其组织在特定的目录（文件夹）中。
- 每个文件由“文件名.扩展名”来标识，扩展名通常有1~3个字母，例如

  “文件名.c”表示C程序的源文件，
  “文件名.cpp”表示C++程序的源文件，
  “文件名.exe”表示可执行文件，
  “文件名.txt”表示文本文件，
  “文件名.jpg”表示一种图像的压缩数据文件，
  “文件名.dat”可以表示自定义数据文件
  ……

- **在C/C++中，文件被看作外存设备中的一段字节串（一般用十六进制数表示），这是一种流式文件处理方式。**

> C家族语言基本延续了这种无结构的流式文件处理方式，与之相对的是有结构的记录式文件处理方式。

- **根据文件中数据存储时的编码可以将C文件分为：**

**（1）二进制文件（binary）**

- 按数据对应的二进制数所组成的字节串存储。例如，对于32位机，整数365可存为00 00 01 6d四个字节串，整数2147483647存为7f ff ff ff四个字节串，字符'A'可存为41一个字节串，字符串"365"可存为33 36 35三个字节串。
- 可以包含任意的二进制字节。
- 一般用于存储无显式结构的数据。

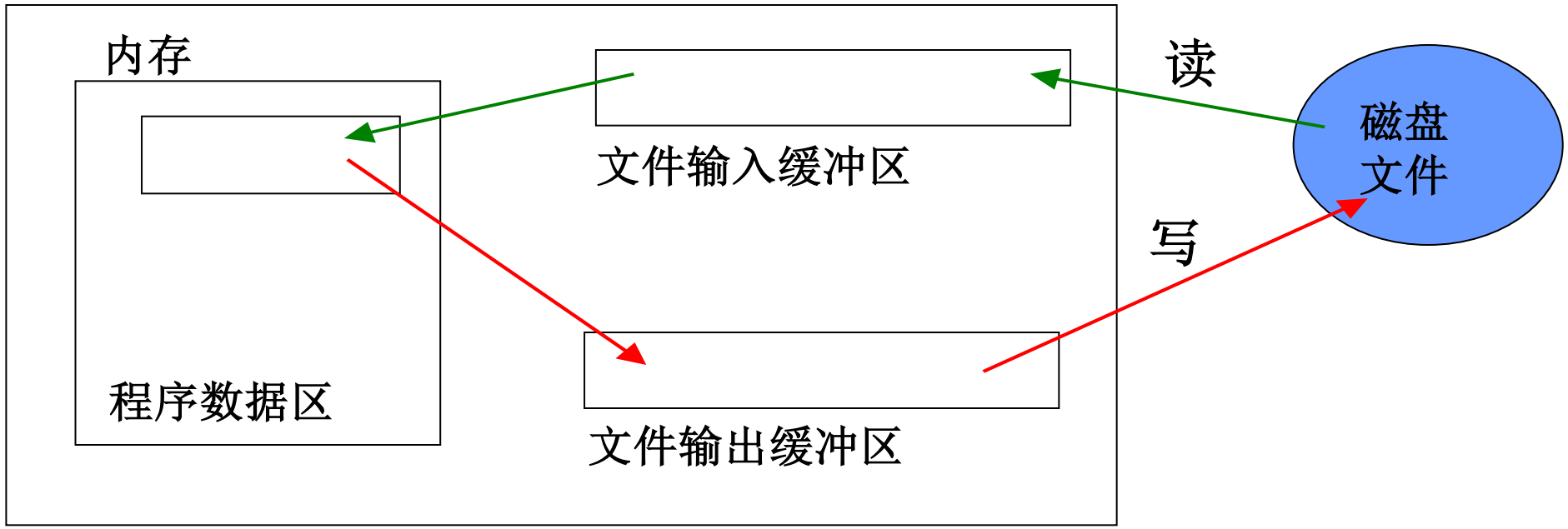
**（2）文本文件（text）**

- 按数据中每个字符的ASCII码组成的字节串存储。例如，整数365存为33 36 35三个字节串，整数2147483647存为32 31 34 37 34 38 33 36 34 37十个字节串，字符'A'存为41一个字节串，字符串"365"存为33 36 35三个字节串。
- 只包含可显示字符和有限的几个控制字符（如：‘\r’、‘\n’、‘\t’等）。
- 一般用于存储具有“行”结构的文本数据。

- 可见，文本文件的平台无关性更好，但文本文件中的数据只能按文本含义来理解，而二进制文件中的数据可以由读/写程序自行约定为各种含义。另外，文本（字符与字符串）在二进制文件和文本文件中一般没有什么不同，但一些特殊字符，例如，表示回车换行的转义字符等，因不同操作系统的处理方式不同会有差别，编程时需注意这个问题（参见后面的**注意事项**）。

- 对文件的操作（即对文件的访问）通常是按字节为单位顺序进行的，包括：
  - 读操作：一般指从外存设备将数据逐个字节读至内存（对于内存而言，是输入数据）
  - 写操作：一般指将内存的数据逐个字节写至外部设备（对于内存而言，是输出数据）

- 对外部设备的访问，速度比内存访问速度低得多，为了减少访问时间，提高程序执行效率，C语言采用缓冲机制，一次读/写一批数据，存于缓冲区，以减少读/写次数。缓冲区的大小由具体的执行环境确定。



# 文件类型指针

- 为了对文件进行有效管理，头文件 stdio.h 中一般定义了一个名为 FILE 的结构类型，例如：

```
typedef struct
{
    short level;                //缓冲区满空程度
    unsigned flags;             //文件状态标志
    char fd;                    //文件描述符
    unsigned char hold;         //无缓冲则不读取字符
    short bsize;                //缓冲区大小
    unsigned char *buffer;      //数据缓冲区
    unsigned char *curp;        //当前位置指针
                                //每读/写一个字节，自动自增1
    short token;                //用于有效性检查
} FILE ;
```

- 对于每一个要操作的文件，都必须定义一个FILE类型的指针变量，并使它指向“文件信息描述区”，以便对文件进行读/写操作。

- “文件信息描述区”由执行环境在程序打开文件时自动创建。

- 文件的打开、读/写操作、关闭等环节需要调用相应的库函数。

# 文件的打开

- 在对文件进行读写操作前，要先打开文件，以便为文件建立“文件信息描述区”，即用程序内部一个表示文件的变量/对象与外部一个具体文件之间建立联系，并指定按文本文件还是按二进制文件来打开。

- 文件的打开是通过库函数fopen实现的，其原型为：

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

  - 参数filename是要打开的文件名（包括路径）；
  - 参数mode是文件的处理模式，它可以是：r、w、a…

```
errno_t fopen_s(FILE **fp, const char *filename, const char *mode);
```

| 打开模式 | 描　　述 |
| --- | --- |
| r | 只读，打开已有文件，不能写 |
| w | 只写，创建或打开，覆盖已有文件 |
| a | 追加，创建或打开，在已有文件末尾追加 |
| r+ | 读写，打开已有文件 |
| w+ | 读写，创建或打开，覆盖已有文件 |
| a+ | 读写，创建或打开，在已有文件末尾追加 |
| t | 按文本方式打开 (缺省) |
| b | 按二进制方式打开 |

- 如果成功打开文件，则函数fopen的返回值为被打开文件信息描述区的地址，否则返回空指针。比如，

```
FILE *fp;
fopen_s(&fp, "d:\\data\\tfile.txt", "w");
```

```
FILE *pfile = fopen("d:\\data\\tfile.txt", "w");
if(pfile == NULL)                          //字符串中的反斜杠需用转义符
  printf("Error! \n");                     在Unix和Linux环境下："d:/data/tfile.txt"
else
  printf("file1.txt has been opened. \n");
```

执行该代码前，用户需先在计算机的d盘建立data目录。该程序执行后，用户可以搜索到相应目录下新创建的tfile.txt文件。
不能成功打开文件的原因有多种，包括当前用户没有磁盘的访问权限、目录不存在等，如果是读模式或读更新模式，文件不存在也会导致文件打开失败。

- 如果成功打开文件，则函数fopen的返回值为被打开文件信息描述区的地址，否则返回空指针。比如，

```
FILE *fp;
fopen_s(&fp, "d:\\data\\tfile.txt", "w");
```

```
FILE *pfile = fopen("d:\\data\\tfile.txt", "w");
if(pfile == NULL)
  printf("Error! \n");
else
  printf("file1.txt has been opened. \n");
```

在Unix和Linux环境下："d:/data/tfile.txt"

执行该代码前，用户需先在计算机的d盘建立data目录。该程序执行后，用户可以搜索到相应目录下新创建的tfile.txt文件。
不能成功打开文件的原因有多种，包括当前用户没有磁盘的访问权限、目录不存在等，如果是读模式或读更新模式，文件不存在也会导致文件打开失败。

# 文件的读/写操作

- 文件的读/写操作也是通过库函数实现的，常用文件操作库函数有

| 函数 | 功能 |
| --- | --- |
| fputc | 输出字符 |
| fgetc | 输入字符 |
| fputs | 输出字符串 |
| fgets | 输入字符串 |

| 函数 | 功能 |
| --- | --- |
| fprintf | 格式化输出 |
| fscanf | 格式化输入 |
| fwrite | 输出数据块 |
| fread | 输入数据块 |

# fputc

- `int fputc(int c, FILE *stream);`
  - 该函数的功能是将字符c写至文件，正常情况下返回字符c的ASCII码，否则（发生写入错误等异常时）返回EOF。

# fprintf

- **int fprintf(FILE *stream, const char *format, ...);**
  - 该函数的功能是将基本类型数据写至文件，正常情况下返回传输字符的个数，否则返回一个负整数。参数format与printf函数的参数类似。例如，

```
pfile = fopen("d:\\data\\tfile.txt", "w");
char name[20];          //学生姓名
int num;                //学号
scanf("%d", &num);
while(num > 0)          //可以输入学号0结束程序
{   scanf("%s", name);
    fprintf(pfile, "%d  %s\n", num, name);
    scanf("%d", &num);
}
```

# fwrite

- `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);`
  - 该函数的功能是按字节将ptr指向的nmemb个字节块的数据写至文件。size_t是在头文件stddef.h中定义的类型，相当于unsigned int。参数size为字节块的大小。正常情况下返回实际写至文件的字节块的个数，异常情况下返回值小于nmemb。如果size或nmemb的值为0，则返回0，文件内容不变。
  - 例如，“fwrite(&i, sizeof(i), 1, pfile);”表示向某二进制文件写入i的值。
  - 又如，“fwrite("\r\n", 1, 2, pfile);”表示向某二进制文件写入两个字符。

# fgetc

- `int fgetc(FILE *stream);`
  - 该函数功能是从文件读取一个字符，正常情况下返回字符的ASCII码，否则返回EOF。

# fscanf //fscanf_s

- **int fscanf(FILE *stream, const char *format, …);**
  - 该函数的功能是从文件读取基本类型数据（文件中的整数、小数之间应有分隔符），正常情况下返回读取数据的个数，否则返回EOF。参数format与scanf函数的参数类似。例如，

```
pfile = fopen("d:\\data\\tfile.txt", "r");
int i = 0;
if(fscanf(pfile, "%d", &i) != EOF)
        //从文件中读取一个整数，赋给变量i
   printf("%d", i);
```

# fread

- `size_t` **`fread`**`(const void *ptr,` **`size`**`, size_t` **`nmemb`**`, FILE *stream);`
  - 该函数的功能是从文件将nmemb个字节块的数据按字节读至ptr所指向的字节块，返回实际读取字节块的个数。例如，“`fread(&i, sizeof(i), 1, pfile);`”表示从某二进制文件中读取数据，赋给变量i。

- 从文件中读取数据时，要根据文件中数据的存储格式选用恰当的库函数，否则无法正确读取数据。一般地，用fgetc、fgets、fscanf和fread函数一一对应fputc、fputs、fprintf和fwrite函数产生的文件数据进行读操作，并且要保持其中参数类型一致。

# 文件的定位

- 一般情况下，文件的读/写操作是顺序进行的，即，在进行读操作时，如果要读文件中的第n个字节，则必须先读前n-1个字节；在进行写操作时，如果要写第n个字节，则也必须先写前n-1个字节。这种文件的访问方式效率往往不高。

- 每个打开的文件都有一个位置指针，指向当前读/写位置，每读/写一个字符，文件的位置指针都会自动往后移动一个位置。文件位置指针还可以用库函数来显式指定

- **int fseek(FILE *stream, long offset, int whence);**
  - 该函数的功能是将文件位置指针指向位置whence + offset，参数whence指出参考位置，其取值可以为SEEK_SET（系统定义的值为0的宏，表示文件头），SEEK_CUR（系统定义的值为1的宏，表示当前位置）或SEEK_END（系统定义的值为2的宏，表示文件末尾），参数offset偏移参考位置的字节数，它可以为正值（向后偏移）或负值（向前偏移）。例如，

```
fseek(pfile, 10, SEEK_SET); //位置指针移至第11个字节处
fseek(pfile, 10, SEEK_CUR); //从当前位置向后移动10个字节
fseek(pfile, -10, SEEK_END);//移至倒数第10个字节处
```

- 文件位置指针的当前位置可以通过库函数获得:

```
long ftell(FILE *stream);  //返回位置指针的位置
```

- 也可以用库函数将文件位置指针拉回到文件头部:

```
void rewind(FILE *stream);
//等价于 fseek(stream, 0, SEEK_SET)
```

# 文件的关闭

- 完成文件的相关操作之后，应及时关闭文件，以释放缓冲区的空间。关闭文件时，执行环境先将文件输出缓冲区的内容都写入文件（无论缓冲区是否为满），然后关闭文件，这样可防止丢失准备写入文件的数据。

- 文件的关闭是通过库函数fclose实现的，fclose函数的原型为：

```
int fclose(FILE *pfile);
```

  - 参数pfile是要关闭的文件信息描述区指针。
  - 如果成功关闭文件，则函数fclose的返回值为0，否则返回EOF。

- 概述
- 文件类型指针
- 文件的打开
- 文件的读写
- 文件的定位
- 文件的关闭
- 注意事项

# EOF

- EOF代表与任何字符的ASCII码都不相同的一个值，是程序中表示文本文件操作异常的宏名，能增强程序的可移植性。开发环境通常在头文件stdio.h中进行类似如下的定义：

```
#define EOF -1
```

- 最典型的文本文件操作异常是从文本文件末尾（end of file）读数据，即读到文本文件结束标志时发生异常。

# 文件结束标志

- 文件结束标志跟操作系统有关，例如，在DOS和Windows环境下，键入Ctrl+Z（ASCII码为0x1A）作为文件的结束标志，在UNIX和Linux环境下，键入Ctrl+D（ASCII码为0x04）作为文件的结束标志

- 例如，库函数getchar将键盘看作输入设备文本文件，当用户从键盘输入文件结束标志时，库函数读到后会返回EOF:

```
char ch;
while((ch = getchar()) != EOF)
   putchar(ch);
```

若将"char"改成"unsigned char"，则当getchar函数返回EOF时，会被隐式转换成无符号数255，从而与此段代码中的EOF（-1）不等，以致即使输入文件结束符循环也无法正常结束。对于char默认为unsigned char的开发环境，应将变量ch定义为"int"，以便能涵盖正常值（ASCII码）与异常值（EOF对应的值）。

# \n

- 类似地，行结束标志也跟操作系统有关。例如，向文本文件写一个字符\n，作为某行的结束：

```
pfile = fopen("d:\\data\\tfile.txt", "w");
fputc('\n', pfile);
```

  - 上述代码在不同环境下执行后效果不同，在Windows环境下查看文件tfile.txt的属性，大小为2字节，而不是1字节。在DOS和Windows环境下，ASCII码为0x0A的回车换行符 \n 写入文本文件时，会自动在前面添加一个ASCII码为0x0D的回车符\r。而在UNIX和Linux环境下，则不会有此现象。

  - 所以，也应注意代码的通用性，通常改为：

```
pfile = fopen("d:\\data\\tfile.txt", "wb");
fputc('\n', pfile);
```

按二进制方式打开

  - 在Windows环境下执行后，文件tfile.txt的大小也为1字节

- **此外，从文本文件读取数据时，对于ASCII码为0x0D的回车符\r + ASCII码为0x0A的回车换行符\n，在DOS和Windows环境下，ASCII码为0x0D的回车符\r会丢弃（在UNIX和Linux环境下则不会丢弃）：**

假定文件中含一个\r和一个\n

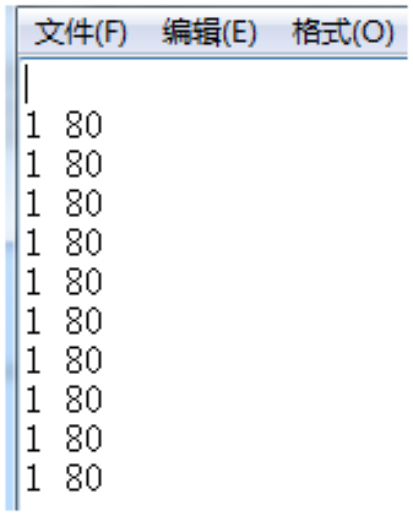
```
char ch;
pfile = fopen("d:\\data\\tfile.txt", "r");
while(fscanf(pfile, "%c", &ch) != EOF)
    printf("%d\n", ch);
```

或while( fread(&ch, 1, 1, pfile) != 0

**在上述代码在Windows下只显示10（回车换行符\n的ASCII码），添加模式字母b，改为按二进制方式打开文件，通常可提高代码的通用性。**

## 例10.1

```
const int N = 10;
struct
{
      int no;
      float score;
} stu[N];
```




```
FILE *fp;
char file[] = "c:\\user\\stuFile.txt";
fopen_s(&fp, file, "r+");
if(!fp)
{
    printf("Open file error!\n");
    exit(0);
}
float s=0;
for(int i = 0; i < N; ++i)
{     fseek(fp, 3, SEEK_CUR);
      fscanf_s(fp, "%f", &stu[i].score);
      s += stu[i].score;
}
printf("%f \n", s);
fclose(fp);
```

# feof函数

- **int feof(FILE *stream);**
  - 该函数的功能是判断文件是否结束，如果文件结束（即文件位置指针在文件末尾）**并继续进行读操作**，则返回非0数，否则返回0。例如，

```
pfile = fopen("d:\\data\\tfile.txt", "r");
int i = 0;
while(!feof(pfile))
{
    if(fscanf(pfile, "%d", &i) != EOF)
        printf("%d\n", i);
}
```

  - 文件结束和其他文件操作异常都有可能导致相关函数返回EOF，feof函数可以专门用来判断文件是否结束（其他错误可以用函数ferror()进行甄别）。

# 注意文件尾部的空白符带来的问题：

```
pfile = fopen("d:\\data\\tfile.txt", "r");
int i = 0;
while(!feof(pfile))
{
    fscanf(pfile, "%d", &i);//读之后文件指针移动
    printf("%d\n", i);
}
```

第二次 fscanf 读一个整数后一切正常，feof 返回 0

第三次 fscanf 读不到一个整数后，i 维持不变，feof 返回 非0

1\n
5\n

输出
1
5
5

# 修改：

```
pfile = fopen("d:\\data\\tfile.txt", "r");
int i = 0;
fscanf(pfile, "%d", &i);//读之后文件指针移动
while(!feof(pfile))
{  printf("%d\n", i);//注意顺序
   fscanf(pfile, "%d", &i);//读之后文件指针移动
}
```

1\n
5\n

输出
1
5

- 或

```
pfile = fopen("d:\\data\\tfile.txt", "r");
int i = 0;
while(!feof(pfile))
{
   if(fscanf(pfile, "%d", &i) != EOF)//读之后文件指针移动
        printf("%d\n", i);
}
```

# 文件尾部无空白符:

```
pfile = fopen("d:\\data\\tfile.txt", "r");
int i = 0;
while(!feof(pfile))
{
    fscanf(pfile, "%d", &i);//读之后文件指针移动
    printf("%d\n", i);
}
```

第二次 fscanf 读一个整数时读不到数据分隔符，feof 返回 非0

1\n
5

输出
1
5

- 或

```
pfile = fopen("d:\\data\\tfile.txt", "r");
int i = 0;
while(!feof(pfile))
{
    if(fscanf(pfile, "%d", &i) != EOF)//读之后文件指针移动
        printf("%d\n", i);
}
```

- 例10.2：带形参的 main 函数

- 一般情况下，程序不需要调用者（比如操作系统）提供参数，定义main函数时不用定义形参，如果程序需要用到调用者提供的参数，则可以在定义main函数时给出形参的定义。

- 一个文件拷贝程序copy.c，可以按“copy file1 file2”的命令形式来执行文件file2至文件file1的拷贝。

```
int main(int argc, char *argv[ ])
{    FILE *fp1, *fp2;
     if( !(fp2 = fopen(argv[2], "r+"))
          || !(fp1 = fopen(argv[1], "w+")) )
                                   //"c:\\user\\file2.txt"
     {    printf("Open file error!\n");
          exit(0);
     }

     int s = 0;
     fscanf(fp2, "%d", &s);
     fprintf(fp1, "%d", s);

     fclose(fp1);
     fclose(fp2);
     return 0;
}
```

# [自学]

- 例10.3 假定无线传感器网络采集的火山口温度值存于数据文件vol.dat中，编程对异常采集数据进行预处理（比如将温度值为0的采集数据用前后相邻两个采集数据的平均值代替，最后一个温度值若为0用前一个数代替）。
- [分析] 可以采用随机数生成的方式模拟数据采集，以便向所创建的文件中写入原始数据。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
const int N = 100;
void DatGenerator(FILE **fpp);
void PreTreat();
```

# [自学]

```
int main( )
{
    FILE *fp;
    char *fname = "d:\\data\\vol.dat";
    if( (fp = fopen(fname,"wb")) == NULL )
                                //打开、建立数据文件
    {     printf("Can't open this file ! \n");
          exit(0);
                                //exit(0)为结束程序运行的库函数
                                //其相关信息在头文件stdlib.h中
    }
    DatGenerator(&fp);    //调用函数，产生原始数据
    fclose(fp);           //关闭文件，保存数据
```

# [自学]

```
    printf("原始数据为: \n");
    FILE *pfile = fopen("d:\\data\\vol.dat", "rb");
    float t;
    while(!feof(pfile))
    {
         if(fread(&t, sizeof(t), 1, pfile) != 0)
         printf("%f\n", t);
    }                    //观察产生的原始数据
    fclose(pfile);

    PreTreat();      //调用函数，预处理数据

    return 0;
}
```

# [自学]

```
void DatGenerator(FILE **fpp)
                        //注意实参是指针变量的地址
{   srand(time(0));
    for(int i = 0; i < N; ++i)
    {
          float j = 100.0*rand()/RAND_MAX;
                        //生成随机数，模拟数据采集
          fwrite(&j, sizeof(j), 1, *fpp);      //写入文件

          float k = 0;
          fwrite(&k, sizeof(k), 1, *fpp);
                        //为便于调试，故意间隔写入异常数据0
    }
}
```

# [自学]

```
void PreTreat()
{  FILE *pfile = fopen("d:\\data\\vol.dat", "r+b");
   float t, tNext, tPre;
   fread(&tPre, sizeof(t), 1, pfile);
                //读取第一个数据作为前一个数据
   fread(&t, sizeof(t), 1, pfile);
                //读取第二个数据作为当前待处理数据
   while(!feof(pfile))
   {    if(fread(&tNext, sizeof(t), 1, pfile) != 0)
                //读取下一个数据
        {    if(t <= 0.1)
             {    t = (tPre + tNext)/2;
                  fseek(pfile, -2*sizeof(t), SEEK_CUR);
                  //位置指针往回移至待处理数据前端
```

# [自学]

```
                    fwrite(&t, sizeof(t), 1, pfile); //改数
                    fseek(pfile, sizeof(t), SEEK_CUR);
                        //恢复位置指针至下一个数据前端
                }
                tPre = t; //当前数据处理完作为下一次处理的前一数
                t = tNext;//下一个数据作为待处理数据
        }
}       //调试阶段可将此循环改为for(int i=0; i < 9; ++i)
fseek(pfile, -sizeof(t), SEEK_CUR);
        //恢复位置指针至最后一个数据前端
fwrite(&tPre, sizeof(t), 1, pfile); //修改最后一个数据
rewind(pfile);
```

# [自学]

```
    printf("\n预处理后的数据为: \n");
    while(!feof(pfile))
    {
        if(fread(&t, sizeof(t), 1, pfile) != 0)
            printf("%f\n", t);
    }    //观察预处理结果数据

    fclose(pfile);
}
```

# 小结

- 程序执行过程中，数据一般以变量、字符串等形式存在，存储在栈中；当数据量比较大或需要长期保存时，则往往存储在**文件**中。（程序的代码也是以文件的形式保存的。另外，C/C++语言把标准输入/输出设备，比如键盘、显示器和打印机，也看成一种文件。）
- 文件类型指针（结构、指针）

- 要求：
  - 了解文件有关的概念和用法
  - 掌握
    - 文件的打开、关闭、读/写等操作方法与注意事项
    - 一个程序代码量≈200行
  - 继续保持良好的编程习惯
    - 文件结束的判断…

- **输入/输出（简称I/O）是程序的一个重要组成部分**
  - 程序运行所需要的数据往往要从内存或外设得到；程序的运行结果通常也要存入内存或外设中去 。
- **分类：**
  - 基于控制台的I/O
    - 从标准输入设备（eg. 键盘）获得数据；把程序结果从标准输出设备（eg. 显示器）输出
  - 基于字符数组的I/O
    - 从程序中的字符数组中获得数据；把程序结果保存到字符数组中
  - 基于文件的I/O
    - 从外存文件获得数据；把程序结果保存到外存文件中
- **在C/C++中，I/O不是语言定义的成分，而是作为标准库的功能由具体的实现（编译器）来提供，有两种途径：**
  - 过程式——I/O库函数（C/C++）
  - 对象式——I/O类库(C++)

# I/O流

- **在C/C++中，I/O操作是一种基于字节流的操作：**
  - 在进行输入操作时，可把输入的数据看成逐个字节地流入到计算机内部（内存）；
  - 在进行输出操作时，则把输出的数据看成逐个字节地从内存流出。

- **在C/C++的标准库中，除了提供基于字节的I/O操作外，为了方便使用，还提供了基于C/C++基本数据类型数据的I/O操作。**

- **在C++程序中也可以对类库中I/O类的一些操作进行重载，使其能对自定义类的对象进行I/O操作。**