第十章 对文件的输入输出

作者：石璞东

参考资料：《C程序设计（第四版）》谭浩强

10.1 C文件的有关知识

10.1.1 什么是文件

在程序设计中，主要用到两种文件：

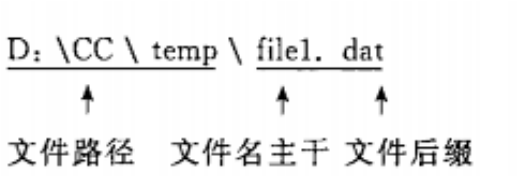
- 程序文件：包括源程序文件`.c`、目标文件`.obj`、可执行文件`.exe`等；
- 数据文件：文件的内容不是程序，而是供程序运行时读写的数据或在程序运行过程中供读入的数据；

10.1.2 文件名

一个文件有一个唯一的文件标识，以便用户识别和引用，文件识别包括3部分：

- 文件路径
- 文件名主干
- 文件名后缀

文件路径表示文件在外部存储设备中的位置，如：



表示`file1.dat`文件存放在`D`盘中的`CC`目录下的`temp`子目录下面，文件标识常被称为文件名，文件名主干的命名规则遵循标识符的命名规则，后缀用来表示文件的性质，如：

- `doc`：`word`生成的文件；
- `txt`：文本文件；
- `dat`：数据文件；
- `c`：`C`语言源程序文件；
- `cpp`：`C++`源程序文件；
- `for`：`FORTRAN`语言源程序文件；
- `pas`：`Pascal`语言源程序文件；
- `obj`：目标文件；
- `exe`：可执行文件；
- `ppt`：电子幻灯片；

- `bmp`：图形文件；

10.1.3 文件的分类

根据数据的组织形式，数据文件可分为`ASCII`文件和二进制文件，数据在内存中是以二进制形式存储的，如果不加转换的输出到外存，就是二进制文件，可以认为它就是存储在内存的数据的映像，即映像文件；如果要求在外存上以`ASCII`代码形式存储，则需要在存储前进行转换，`ASCII`文件又称文本文件，每一个字节存放一个字符的`ASCII`代码。

一个数据在磁盘上怎样存储呢？字符一律以`ASCII`形式存储，数值型数据既可以用`ASCII`形式存储，也可以用二进制形式存储。

10.1.4 文件缓冲区

所谓缓冲文件系统是指系统自动地在内存区为程序中每一个正在使用的文件开辟一个文件缓冲区，从内存向磁盘输出数据必须先送到内存中的缓冲区，装满缓冲区后才一起送到磁盘去，如果从磁盘向计算机读入数据，则一次从磁盘文件将一批数据输入到内存缓冲区，然后在从缓冲区逐个地将数据送到程序数据区，缓冲区的大小由各个具体的`C`编译系统确定。

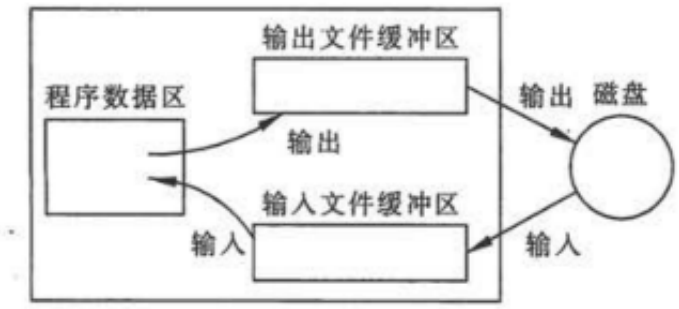


图 10.2

10.1.5 文件类型指针

每个被使用的文件都在内存中开辟一个相应的文件信息区，用来存放文件的有关信息，如文件的名字、文件状态以及文件当前位置等，如`FILE *fp`所示即为定义`fp`是一个指向`FILE`类型数据的指针变量，可以使`fp`指向某一个文件的文件信息区，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。

指向文件的指针变量并不是指向外部介质上的数据文件的开头，而是指向内存中的文件信息区的开头。

10.2 打开与关闭文件

所谓**打开**是指为文件建立相应的信息区（用来存放有关文件的信息）和文件缓冲区（用来暂时存放输入输出的数据），在打开文件的同时，一般都指定一个指针变量指向该文件，也就是建立起指针变量与文件之间的联系，这样就可以通过该指针变量对文件进行读写了，所谓**关闭**是指撤销文件信息区和文件缓冲区，使文件指针变量不再指向该文件，显然就无法进行对文件的读写了。

10.2.1 用`fopen`函数打开数据

例10.1 文件的打开与关闭

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

int main(){
    FILE *input ,*destfile;
    int t;
    input = fopen("1.txt","r");
    destfile = fopen("2.txt" , "w");
    if(input == NULL){
        // 抛出最近一次的系统错误信息
        perror("the file is empity");
        exit(EXIT_FAILURE);
    }
    else{
        perror("the file is not empity");
        // feof()函数用来检测当前流文件上的文件结束表识，判断是否读到了文件结尾。
        while(!feof(input)){
            t = fgetc(input);
            if(t =='{'){
                t ='s';
            }
            printf("%c" , t);
            fputc(t , destfile);
        }

    }
    // 用于清空文件缓冲区
    fflush(stdout);
    fclose(destfile);
    fclose(input);
    return 0;
}
```

`fopen`函数的调用方式为：`fopen(文件名，使用文件方式)`

模式	描述
r	打开一个已有的文本文件，允许读取文件。
w	打开一个文本文件，允许写入文件。如果文件不存在，则会创建一个新文件。在这里，您的程序会从文件的开头写入内容。如果文件存在，则该会被截断为零长度，重新写入。
a	打开一个文本文件，以追加模式写入文件。如果文件不存在，则会创建一个新文件。在这里，您的程序会在已有的文件内容中追加内容。
r+	打开一个文本文件，允许读写文件。
w+	打开一个文本文件，允许读写文件。如果文件已存在，则文件会被截断为零长度，如果文件不存在，则会创建一个新文件。
a+	打开一个文本文件，允许读写文件。如果文件不存在，则会创建一个新文件。读取会从文件的开头开始，写入则只能是追加模式。

程序中可以使用3个标准的流文件 — **标准输入流`stdin`**、**标准输出流`stdout`**、**标准出错输出流`stderr`**，系统已对这3个文件指定了与终端的对应关系，标准输入流是从终端的输入，标准输出流是向终端的输出，标准出错输出流是当程序出错时将出错信息发送到终端。

程序开始运行时系统自动打开这3个标准流文件，因此，程序编写者不需要在程序中用`fopen`函数打开它们。所以我们以前用到的从终端输入或输出到终端都不需要打开终端文件，系统定义了3个文件指针变量`stdin`、`stdout`、`stderr`，分别指向标准输入流、标准输出流和标准出错输出流，可以通过这3个指针变量对以上3种流进行操作，它们都以终端作为输入输出对象，如果程序中指定要从`stdin`所指的文件输入数据，就是指从终端键盘输入数据。

10.2.2 用`fclose`函数关闭数据文件

`fclose`函数调用的一般形式为：`fclose(文件指针)`

在使用完一个文件后应该关闭它，以防止它再被误用，**关闭**就是撤销文件信息区和文件缓冲区，使文件指针变量不再指向该文件，也就是文件指针变量与文件脱钩，此后不能再通过该指针对原来与其相联系的文件进行读写操作，除非再次打开，使该指针变量重新指向该文件。

如果不关闭文件将会丢失数据。因为，在向文件写数据时，是先将数据输出到缓冲区，待缓冲区充满后才正式输出给文件。如果当数据未充满缓冲区而程序结束运行，就有可能使缓冲区中的数据丢失。要用`fclose`函数关闭文件，先把缓冲区中的数据输出到磁盘文件，然后才撤销文件信息区。有的编译系统在程序结束前会自动先将缓冲区中的数据写到文件，从而避免了这个问题，但还是应当养成在程序终止之前关闭所有文件的习惯。

`fclose`函数也带回一个值，当成功地执行了关闭操作，则返回值为0；否则返回`EOF(-1)`。

10.3 顺序读写数据文件

10.3.1 怎样向文件读写字符

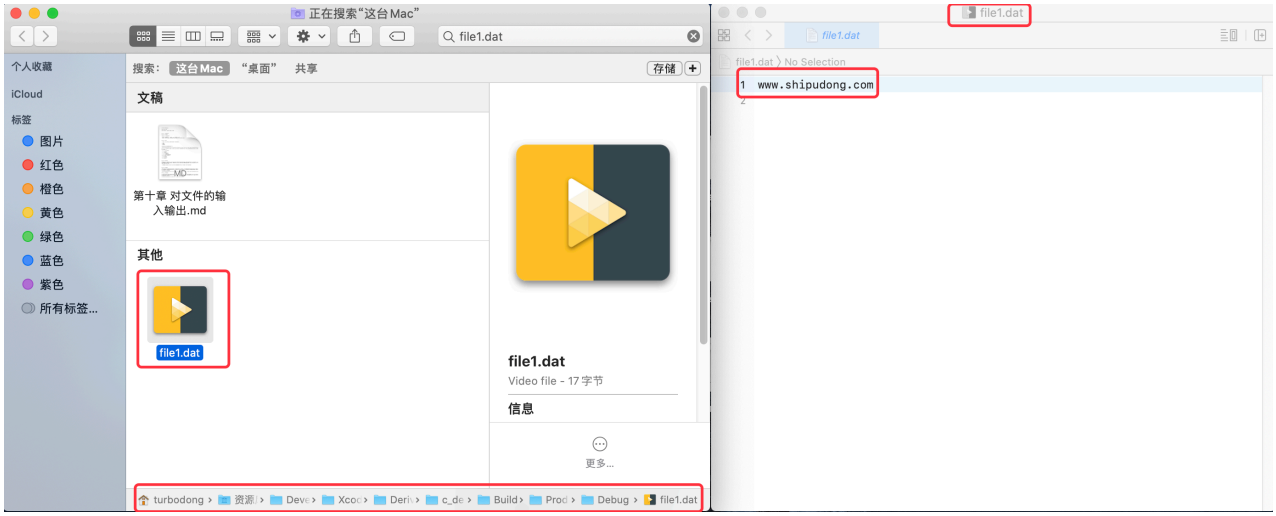
表 10.2　读写一个字符的函数

函数名	调用形式	功　能	返　回　值
fgetc	fgetc(fp)	从 fp 指向的文件读入一个字符	读成功，带回所读的字符，失败则返回文件结束标志 EOF(即－1)
fputc	fputc(ch,fp)	把字符 ch 写到文件指针变量 fp 所指向的文件中	输出成功，返回值就是输出的字符；输出失败，则返回 EOF(即－1)

例10.2 从键盘输入字符，逐个把它们送到磁盘上去，直到用户输入一个`#`为止。

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

int main(){
    FILE *fp;
    char ch,filename[10];
    printf("请输入所用的文件名：");
    scanf("%s",filename);
    if ((fp = fopen(filename,"w")) == NULL) {
        printf("无法打开此文件！\n");
        exit(0);
    }
    ch = getchar();
    printf("请输入一个字符（以#结束）：");
    ch = getchar();
    while (ch!='#')
    {
        fputc(ch,fp);
        putchar(ch);
        ch = getchar();
    }
    fclose(fp);
    putchar(10);
    return 0;
}
```

效果如下所示：

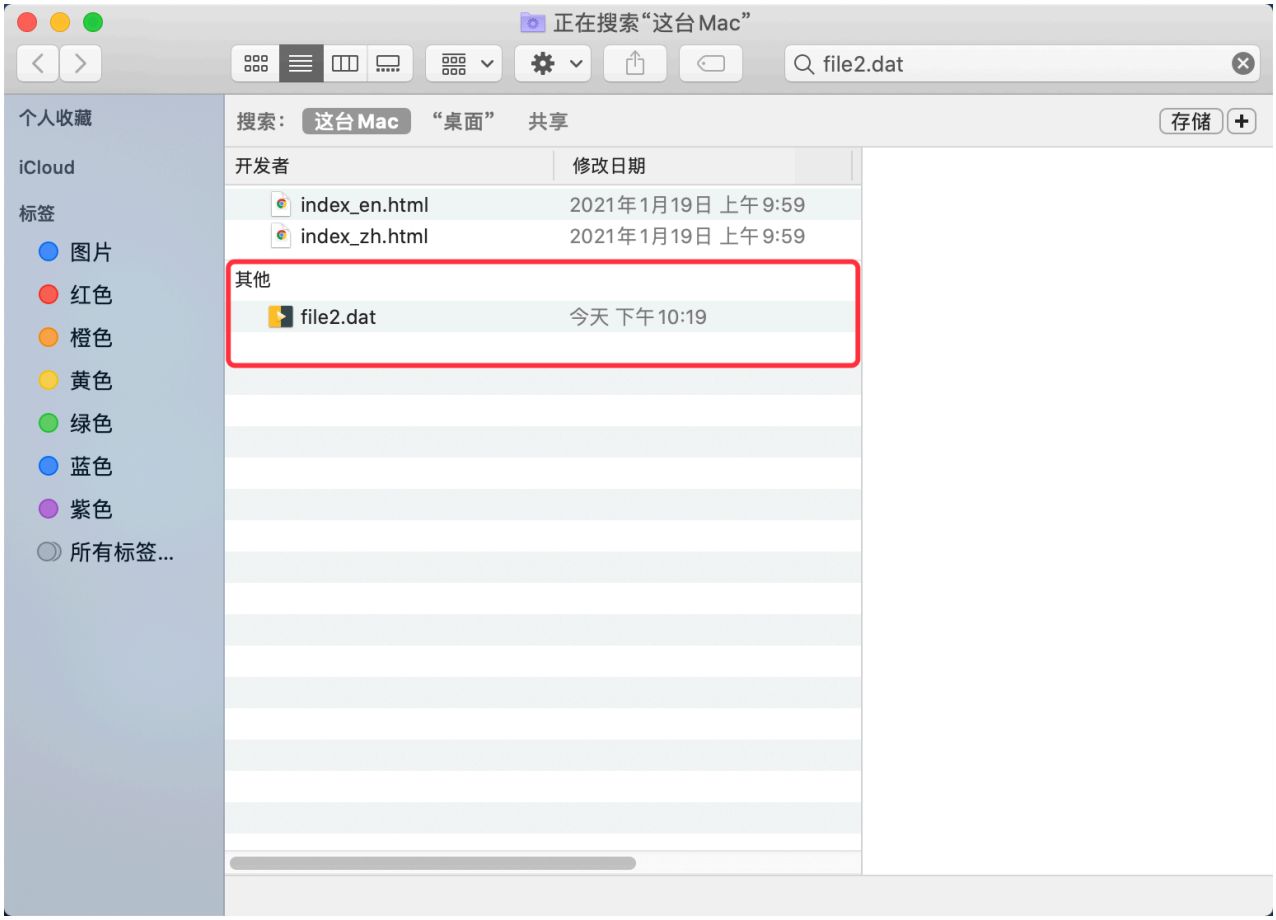


`fopen`函数打开成功之后会返回该文件所建立的信息区的起始地址，并将其赋值给指针变量`fp`；`exit`是标准的`C`库函数，其作用是使程序终止，用此函数时在程序的开头应包含`stdlib.h`头文件。

例10.3 将`file1.dat`的内容复制到`file2.data`中。

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

int main(){
    FILE *in,*out;
    char ch,infile[10],outfile[10];
    printf("请输入读入的文件名：");
    scanf("%s",infile);
    printf("请输入要写入的文件的名字：");
    scanf("%s",outfile);
    if ((in = fopen(infile,"r")) == NULL) {
        printf("无法打开此文件！\n");
        exit(0);
    }
    if((out = fopen(outfile, "w"))==NULL){
        printf("无法打开此文件！\n");
        exit(0);
    }
    while (!feof(in)) {
        ch = fgetc(in);
        fputc(ch, out);
        putchar(ch);
    }

    fclose(in);
    fclose(out);
    putchar(10);
    return 0;
}
```

效果如下所示：



用`feof`函数可以检查到文件读写位置标记是否移到文件的末尾，即磁盘文件是否结束，若结束则返回1，否则返回0。

10.3.2 怎样向文件读写一个字符串

表 10.3　读写一个字符串的函数

函数名	调用形式	功　能	返　回　值
fgets	fgets(str,n,fp)	从 fp 指向的文件读入一个长度为(n－1)的字符串，存放到字符数组 str 中。	读成功，返回地址 str，失败则返回 NULL
fputs	fputs(str,fp)	把 str 所指向的字符串写到文件指针变量 fp 所指向的文件中	输出成功，返回 0；否则返回非 0 值

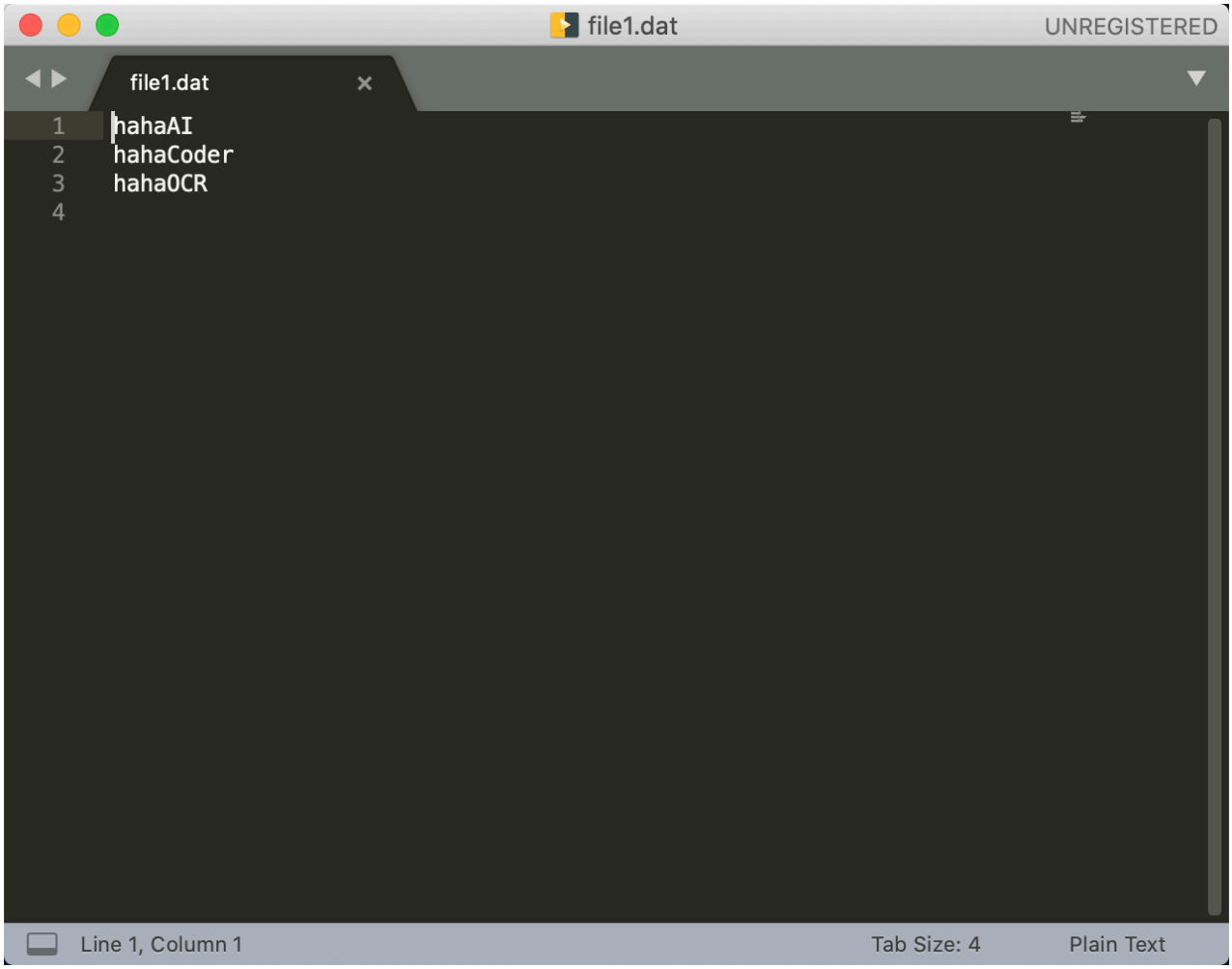
「注」：

- 若`fgets`函数执行成功，则返回值为`str`数组首元素的地址，如果一开始就遇到文件尾或读数据出错，则返回`NULL`；
- `fputs`函数的原型为`int fputs(char *str,FILE *fp)`，其作用是将`str`所指向的字符串输出到`fp`所指向的文件中，该函数第一个参数可以是字符串常量、字符数组名或字符型指针，字符串末尾的`\0`不输出，若输出成功，函数值为0，失败时，函数值为`EOF`；

例10.4 从键盘读入若干个字符串，对它们按字母大小的顺序排序，然后将排好序的字符串送入磁盘文件中保存。

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

int main(){
    FILE *fp;
    char str[3][10],temp[10];
    int i,j,k,n=3;
    printf("请输入字符串：");
    for (i = 0; i < 3; i++) {
        gets(str[i]);
    }
    for (i = 0; i < n-1; i++) {
        k = i;
        for (j = i+1; j < n; j++) {
            if (strcmp(str[k], str[j])>0) {
                k = j;
            }
        }
        if(k!=i){
            strcpy(temp, str[i]);
            strcpy(str[i], str[k]);
            strcpy(str[k], temp);
        }
    }
    if((fp = fopen("file1.dat", "w"))==NULL){
        printf("不能打开文件！\n");
        exit(0);
    }
    printf("排序之后的字符如下：\n");
    for (i = 0; i < n; i++) {
        fputs(str[i], fp);
        fputs("\n", fp);
        printf("%s\n",str[i]);
    }
    return 0;
}
```

效果如下所示：



接下来，我们通过程序读`file1.dat`中的内容，请看代码示例：

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

int main(){
    FILE *fp;
    char str[3][10];
    int i = 0;
    if((fp = fopen("file1.dat", "r"))==NULL){
        printf("不能打开文件! \n");
        exit(0);
    }
    while (fgets(str[i], 10, fp)!=NULL) {
        printf("%s",str[i]);
        i++;
    }
    fclose(fp);
    return 0;
}
```

10.3.3 用格式化的方法读写文件

`fprintf`和`fscanf`函数的读写对象不是终端而是文件，它们的一般调用方式为：

- `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)`

用以上两个函数对磁盘文件读写较为方便，但由于在输入时要将文件中的`ASCII`码转换为二进制形式再保存在内存变量中，再输出时又要将内存中的二进制形式转换成字符，即内存与磁盘的交换数据次数过于频繁。

10.3.4 用二进制方式向文件读写一组数据

`fread`函数和`fwrite`函数的一般调用形式为：`fread(buffer,size,count,fp)`、`fwrite(buffer,size,count,fp)`，其中：

- `buffer`：是一个地址，对`fread`来说，它是用来存放从文件读入的数据的存储区的地址，对`fwrite`来说，是要把此地址开始的存储区中的数据向文件输出；
- `size`：要读写的字节数；
- `count`：要读写多少个数据项；
- `fp`：`FILE`类型指针；

以下是菜鸟教程中关于`fread`函数和`fwrite`函数的介绍：

- `fread`函数

声明

下面是 fread() 函数的声明。

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

参数

- **ptr** -- 这是指向带有最小尺寸 *size*nmemb* 字节的内存块的指针。
- **size** -- 这是要读取的每个元素的大小，以字节为单位。
- **nmemb** -- 这是元素的个数，每个元素的大小为 size 字节。
- **stream** -- 这是指向 FILE 对象的指针，该 FILE 对象指定了一个输入流。

返回值

成功读取的元素总数会以 size_t 对象返回，size_t 对象是一个整型数据类型。如果总数与 nmemb 参数不同，则可能发生了一个错误或者到达了文件末尾。

- `fwrite`函数

C 库函数 **size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)** 把 **ptr** 所指向的数组中的数据写入到给定流 **stream** 中。

声明

下面是 fwrite() 函数的声明。

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

参数

- **ptr** -- 这是指向要被写入的元素数组的指针。
- **size** -- 这是要被写入的每个元素的大小，以字节为单位。
- **nmemb** -- 这是元素的个数，每个元素的大小为 size 字节。
- **stream** -- 这是指向 FILE 对象的指针，该 FILE 对象指定了一个输出流。

返回值

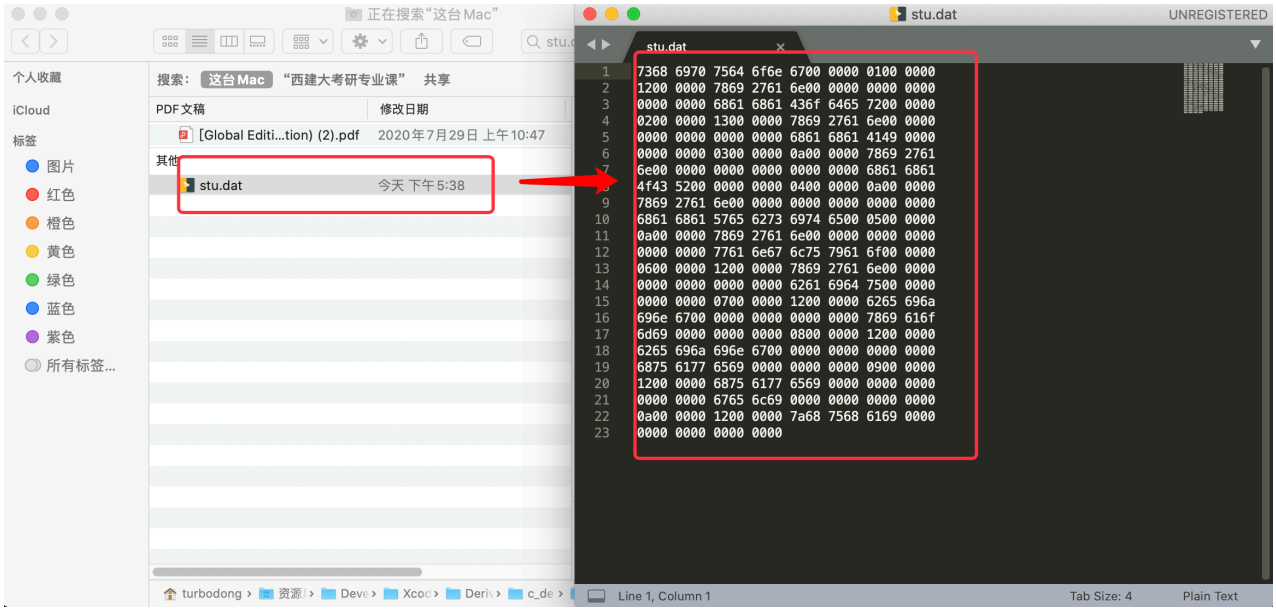
如果成功，该函数返回一个 size_t 对象，表示元素的总数，该对象是一个整型数据类型。如果该数字与 nmemb 参数不同，则会显示一个错误。

例10.5 从键盘输入10个学生的有关数据，然后把它们转存到磁盘文件上去。

```c
#include <stdio.h>
#include<stdlib.h>
#define SIZE 10

struct Student_type{
    char name[10];
    int num;
    int age;
    char addr[15];
}stud[SIZE];

void save(){
    FILE *fp;
    if((fp = fopen("stu.dat", "wb"))==NULL){
        printf("不能打开文件! \n");
        exit(0);
    }
    for (int i = 0; i < SIZE; i++) {
        if (fwrite(&stud[i], sizeof(struct Student_type), 1, fp)!=1) {
            printf("写入文件错误! \n");
        }
    }
    fclose(fp);
}

void read(){
    FILE *fp;
    if((fp = fopen("stu.dat", "rb"))==NULL){
        printf("不能打开文件! \n");
        exit(0);
```

```
    }
    for (int i = 0; i < SIZE; i++) {
        fread(&stud[i], sizeof(struct Student_type), 1, fp);
        printf("姓名：%s\t学号：%d\t年龄：%d\t地址：
%s\n",stud[i].name,stud[i].num,stud[i].age,stud[i].addr);
    }
    fclose(fp);
}

int main(){
    printf("请输入学生数据：\n");
    for (int i = 0; i < SIZE; i++) {
        scanf("%s %d %d
%s",stud[i].name,&stud[i].num,&stud[i].age,stud[i].addr);
    }
    save();
    read();
    return 0;
}
```

效果如下所示：



接着，我们从磁盘中读取信息并将其打印出来，效果如下所示：

```
    if((fp = fopen("stu.dat","rb"))==NULL){
        printf("不能打开文件! \n");
        exit(0);
    }
    for (int i = 0; i < SIZE; i++) {
        fread(&stud[i], sizeof(struct Student_type), 1, fp);
        printf("姓名: %s\t学号: %d\t年龄: %d\t地址: %s\n",stud[i].name,stud[i].num,stud[i].age
    }
    fclose(fp);
}

int main(){
    printf("请输入学生数据: \n");
    for (int i = 0; i < SIZE; i++) {
        scanf("%s %d %d %s",stud[i].name,&stud[i].num,&stud[i].age,stud[i].addr);
    }
    save();
    read();
    return 0;
```

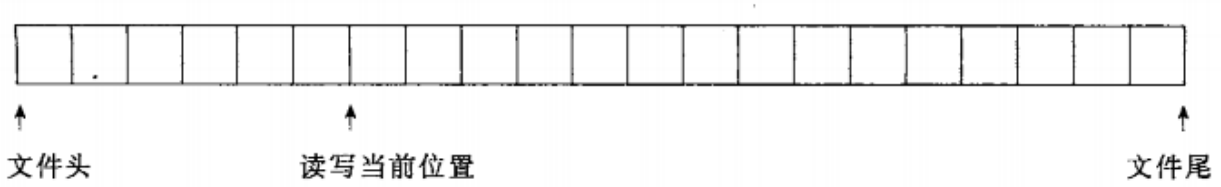
```
请输入学生数据:
grandmother 1 18 xi'an
father 2 18 xi'an
mother 3 18 xi'an
sister 4 18 xi'an
shipudong 5 18 xi'an
wangluyao 6 18 xi'an
cousin 7 18 xi'an
friends 8 18 xi'an
teachers 9 18 xi'an
stranger 10 18 xi'an
姓名: grandmother学号: 1   年龄: 18 地址: xi'an
姓名: father 学号: 2   年龄: 18 地址: xi'an
姓名: mother 学号: 3   年龄: 18 地址: xi'an
姓名: sister 学号: 4   年龄: 18 地址: xi'an
姓名: shipudong  学号: 5   年龄: 18 地址: xi'an
姓名: wangluyao  学号: 6   年龄: 18 地址: xi'an
姓名: cousin 学号: 7   年龄: 18 地址: xi'an
姓名: friends学号: 8   年龄: 18 地址: xi'an
姓名: teachers   学号: 9   年龄: 18 地址: xi'an
姓名: stranger   学号: 10 年龄: 18 地址: xi'an
Program ended with exit code: 0
```

10.4 随机读写数据文件

假设需要查询几百万人中最后一个人的资料，按照顺序读写的方式，则需要将前面所有人查询完成之后才可以读取；随机访问不是按数据在文件中的物理位置次序进行读写，而是可以对任何位置上的数据进行访问，显然这种方法比顺序访问效率高得多。

10.4.1 文件位置标记及其定位

- 文件位置标记



对流式文件既可以进行顺序读写，也可以进行随机读写，关键在于控制文件的位置标记。如果文件位置标记是按字节位置顺序移动的，就是顺序读写。如果能将文件位置标记按需移动到任意位置，就可以实现随机读写。

所谓随机读写，是指读写完上一个字符/字节后，并不一定要读写其后续的字符/字节，而可以读写文件中任意位置上所需要的字符，即对文件读写数据的顺序和数据在文件中的物理顺序一般是不一致的，可以在任何位置写入数据，在任何位置读取数据。

- 文件位置标定的定位

我们可以通过`rewind`函数使文件位置标记指向文件开头，该函数没有返回值；还可以通过`fseek`函数改变文件位置标记，其调用形式为`fseek(文件类型指针，位移量，起始点)`，起始点用0、1、2代替，0代表文件开始位置，1代表当前位置，2代表文件末尾位置。

表 10.4

起 始 点	名 字	用数字代表
文件开始位置	SEEK_SET	0
文件当前位置	SEEK_CUR	1
文件末尾位置	SEEK_END	2

位移量指以起始点为基点向前移动的字节数，应是`long`型数据，即在数字的末尾加一个字母`L`。

- `fseek(fp,100L,0)`：将文件位置标记向前移到距离文件开头100个字节处；
- `fseek(fp,50L,1)`：将文件位置标记向前移到距离当前位置50个字节处；
- `fseek(fp,-10L,2)`：将文件位置标记从文件末尾处向后退10个字节；

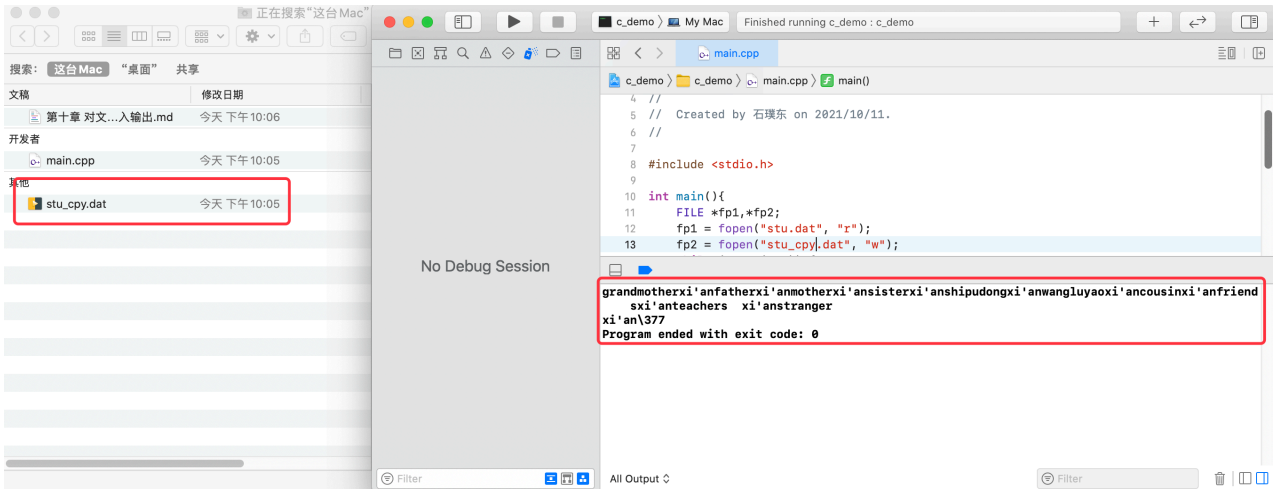
我们还可以通过`ftell`函数测定文件位置标记的当前位置，由于文件中的文件位置标记经常移动，人们往往不容易知道其当前位置，所以常用`ftell`函数的到当前位置，用相对于文件开头的位移量来表示，如果调用函数时出错，`ftell`函数返回值为`-1L`。

例10.6 将磁盘文件的信息第1次显示在屏幕上，第2次把它复制到另一文件上。

```c
#include <stdio.h>

int main(){
    FILE *fp1,*fp2;
    fp1 = fopen("stu.dat", "r");
    fp2 = fopen("stu_cpy.dat", "w");
    while (!feof(fp1)) {
        putchar(getc(fp1));
    }
```

```
    putchar(10);
    rewind(fp1);
    while (!feof(fp1)) {
        putc(getc(fp1), fp2);
    }
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

请看效果展示：



10.4.2 随机读写

例10.7 在磁盘文件上存有10个学生的数据，要求将第1、3、5、7、9个学生数据输入计算机，并在屏幕上显示出来。

```
#include <stdio.h>
#include<stdlib.h>
#define SIZE 10

struct Student_type{
    char name[10];
    int num;
    int age;
    char addr[15];
}stud[SIZE];

int main(){
    FILE *fp;
    if((fp = fopen("stu.dat","rb"))==NULL){
        printf("不能打开文件！\n");
        exit(0);
    }
```

```
    for (int i = 0; i < SIZE; i+=2) {
        fseek(fp, i * sizeof(struct Student_type), 0);
        fread(&stud[i], sizeof(struct Student_type), 1, fp);
        printf("名字：%5s\t学号：%d\t年龄：%d\t地址：
%s\n",stud[i].name,stud[i].num,stud[i].age,stud[i].addr);
    }
    fclose(fp);
    return 0;
}
```

效果如下所示：

```
int main(){
    FILE *fp;
    if((fp = fopen("stu.dat","rb"))==NULL){
        printf("不能打开文件！\n");
        exit(0);
    }
    for (int i = 0; i < SIZE; i+=2) {
        fseek(fp, i * sizeof(struct Student_type), 0);
        fread(&stud[i], sizeof(struct Student_type), 1, fp);
        printf("名字：%5s\t学号：%d\t年龄：%d\t地址：%s\n",stud[i].name,stud[i].num,stud[i].age,stud[i].addr);
    }
```

```
名字：grandmother学号：1   年龄：18 地址：xi'an
名字：mother 学号：3   年龄：18 地址：xi'an
名字：shipudong   学号：5   年龄：18 地址：xi'an
名字：cousin 学号：7   年龄：18 地址：xi'an
名字：teachers    学号：9   年龄：18 地址：xi'an
Program ended with exit code: 0
```

10.5 文件读写的出错检测

- `ferror`函数

我们可以在调用各种输入输出函数（如`putc`、`getc`、`fread`、`fwrite`等）时，通过`ferror`函数进行检查，其一般调用形式为`ferror(fp);`，若其返回值为0，则表示未出错；若其返回值为非零值，则表示出错。

在执行`fopen`函数时，`ferror`函数的初始值自动置为0。

- `clearerr`函数

`clearerr`的作用是使文件错误标志和文件结束标志置为0，假设在调用一个输入输出函数时出现错误，`ferror`函数值为一个非零值，应该立即调用`clearerr(fp)`，使`ferror(fp)`的值变为0，以便再进行下一次的检测。

课后题：3、5、11