

第10章 对文件的输入输出（2）



复习回顾

➤ 上次课的内容：

◆ 文件的基本概念

◆ 数据的存储方式

◆ 文件的打开关闭

◆ 文本文件顺序读写

◆ “文件的故事”：卡耐基-梅隆（CMU）大学的编程竞赛

参赛者必须读入一个文件（文件的内容是一些数值），并打印这些数值的平均数。只有两个规则：

1. 程序必须用Pascal或C编写，可以上交多个作品。
2. 程序的运行速度要尽可能地快，**运行时间最短的程序将获得优胜！**

文件读取：猜想中的赢家

➤ 采取代码优化措施：

◆ 消除循环

◆ 函数代码就地扩展

◆ 公共子表达式消除

◆ 改进寄存器分配

◆ 省略运行时对数组边界检查

◆ 操作符长度削减（把指数操作转为乘法，把乘法转为移位操作或加法操作）

◆

数据文件大约包含10000个数值，
当时读入和处理每个数需要一毫秒，
所以预测最快的程序也要用10秒。

季军的做法：优化的极致

- **殚精竭虑，用优化机器代码来解决问题**
 - ◆ **把指令作为整型数组存储在程序中**
 - ◆ **在程序中覆盖堆栈上的返回地址**
 - ◆ **使程序跳转到这个整型数组并逐条执行这些指令**
 - ◆ **结果该作品所花时间比预想的10秒的最短时间还要少一点**

亚军的做法：狡猾的投机

➤ 利用竞争规则而不是怪异的编码

◆ 提交两个不同的程序

- 一个程序读入数据，用正常的方法计算平均值，并将答案写入一个文件
- 另一个程序绝大部分时间处于睡眠状态，每个几秒醒来一次检查答案文件是否已经存在，如果存在，就打印结果

◆ 第二个程序只需要占用**几毫秒**的CPU时间，所以凭借此程序登上亚军宝座

冠军的做法：颠覆者为王

➤ 充分利用操作系统的漏洞

- ◆ 找到进程控制块相对于堆栈底部的存储位置
- ◆ 用一个指针来访问进程控制块，并用一个非常大的值覆盖“CPU已使用时间”这个字段
- ◆ 操作系统未曾想到CPU时间会有如此之大，因此以二进制补码把这个数解释为负数！
- ◆ 冠军作品的最终成绩是 **-3 秒**

比赛总结

- 循规蹈矩的编程牛人拼不过投机分子
- 技术含量越高的作弊越有效果
- 既有编程技术又会利用规则才是无敌的
- 不管怎样，学好编程技术会使你更强大
- btw，由于引发争议，该比赛不再举行。

用格式化的方式读写文本文件

➤ 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);`

如：

`fprintf (fp, "%d, %6.2f", i, f);`

`fscanf (fp, "%d, %f", &i, &f);`

格式化方式读写文本的优缺点

➤ 优点

◆与printf和scanf相仿，使用方便，容易理解


➤ 缺点

◆输入时需要将文件中的ASCII码转为二进制形式保存在内存，输出时需要将内存中的二进制形式转换成字符，要花费较多时间

用fscanf读取文本文件内容

```
1. #include <stdio.h>
2. #include <stdlib.h>

3. int main()
4. {
5.     FILE * fp;
6.     int i; float f;
7.     fp=fopen("c:\\num.txt", "r"); //“文本只读” 方式打开文件
8.     if (fp==NULL)
9.         exit(0);
10.    fscanf(fp, "%d", &i);
11.    printf("%d\n", i);
12.    fscanf(fp, "%f", &f);
13.    printf("%f\n", f);
14.    fclose(fp); //关闭文件
15.
16.    return 0;
17. }
```



The diagram illustrates the state of a file after the first `fscanf` call. A black box displays the values `3` and `0.140000`. Below it, a horizontal sequence of boxes represents the file's content: `3`, `.`, `1`, `4`, and `EOF`. A large black arrow labeled `fp` points to the first box (`3`). Three red arrows point upwards to the first, second, and fifth boxes, with a long white arrow pointing from the first to the second box. Below these arrows, the text **文件位置标记** (File Position Marker) is written in red.

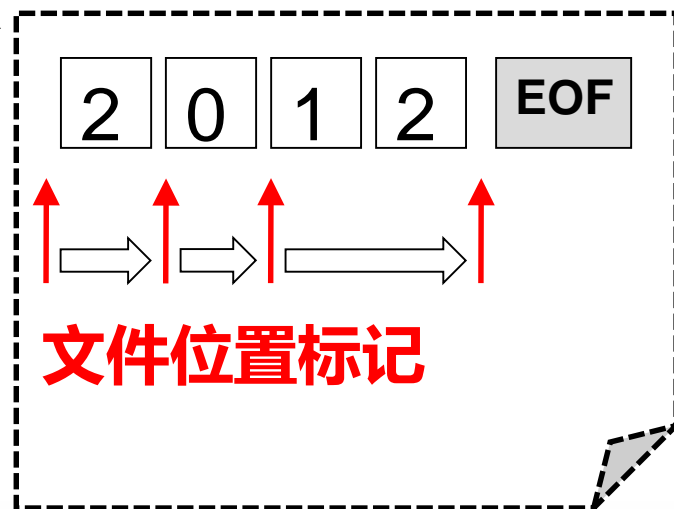
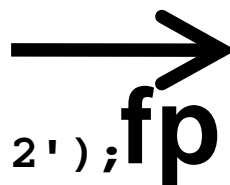
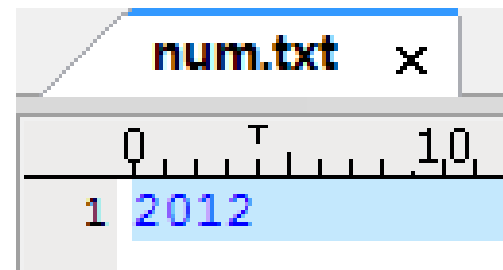
把各种类型数据写入文本文件

```
1. #include <stdio.h>
2. #include <stdlib.h>

3. int main()
4. {
5.     FILE * fp;
6.     fp=fopen("c:\\num.txt", "w"); //“文本只读” 方式打开文件
7.     if (fp==NULL)
8.         exit(0);

9.     fprintf(fp, "%c", '2'); fp
10.    fprintf(fp, "%d", 0);
11.    fprintf(fp, "%s", "12");

12.    fclose(fp); //关闭文件
13.
14.    return 0;
15. }
```



用二进制方式读写文件

➤ 读写一组数据块的函数fread与fwrite:

`int fread(void* buffer, int size, int count, FILE* pfile);`

`int fwrite(void* buffer, int size, int count, FILE* pfile);`

- ◆ **参数buffer**是一个指针，用来指向一个数据内存区域，表示“读入/输出”数据在内存中的起始地址
- ◆ **参数size**表示要进行读/写的每个数据项的字节数
- ◆ **参数count**表示将要读/写多少个size字节的数据项
- ◆ **参数pfile**是FILE类型指针，如果文件以二进制的形式打开，那么用fread()和fwrite()可以读/写任何类型的信息，无论是数组还是结构体。

fread的调用说明

➤ 一般调用形式为：

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

- ◆ 执行fread()函数时，**文件位置指针**会随着数据的读取而向后移动，最后移动结束的位置等于实际读出的字节数。
- ◆ 该函数执行结束后，将**返回**实际读出的数据元素个数，这个“个数”不一定等于设置的count，因为若文件中没有足够的数据元素项或读取过程中出错，都会导致返回的“个数”小于count。

fwrite的调用说明

➤ 一般调用形式为：

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

- ◆ `fwrite()` 函数从 `buffer` 指针指向的内存区中取出长度为 `count` 个 `size` 字节的数据元素项，写入到指针 `pfile` 所指的文件中。
- ◆ 执行该操作后，**文件位置指针** 将向后移动，移动的字节数等于写入文件的字节数目。
- ◆ 该函数操作完成后，**返回** 成功写入的数据元素个数。

创建二进制文件并写入一个数组

```
1. #include <stdio.h>
2. #include <stdlib.h>
```

```
3. int main()
```

```
4. {
```

```
5.     FILE * fp;
```

```
6.     float arr[2]={20, 12};
```

```
7.     fp = fopen("c:\\num.bin", "wb"); //“文本只写” 方式打开文件
```

```
8.     if (fp==NULL)
```

```
9.         exit(0);
```

```
10.    fwrite(arr, sizeof(float),
```

```
11.           2, fp);
```

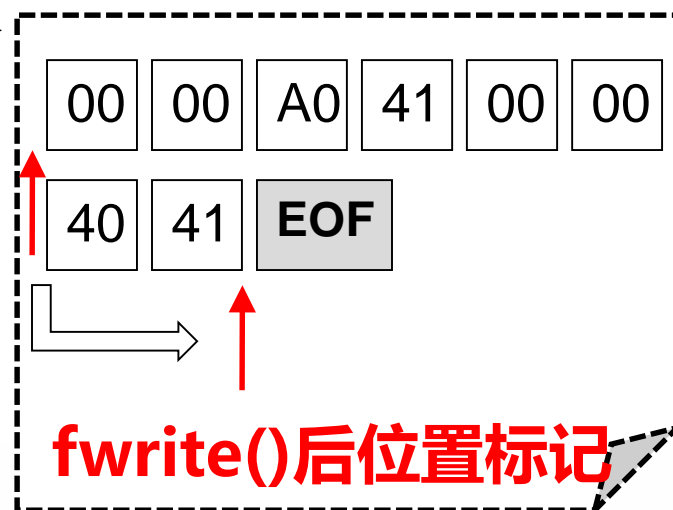
```
12.    fclose(fp); //关闭文件
```

```
13.    return 0;
```

```
14. }
```

num.bin x									
		0	1	2	3	4	5	6	7
00000000h:		00	00	A0	41	00	00	40	41

→
fp



从二进制文件读取内容存入数组

```
1. #include <stdio.h>
2. #include <stdlib.h>
```

```
3. int main()
```

```
4. {
```

```
5.     FILE * fp;
```

```
6.     float arr[2];
```

```
7.     int i;
```

```
8.     fp = fopen("c:\\num.bin", "rb"); //“文本只读” 方式打开文件
```

```
9.     if (fp==NULL)
```

```
10.         exit(0);
```

```
11.     fread(arr, sizeof(float), fp
```

```
12.         2, fp);
```

```
13.     printf("%f\n", arr[0]);
```

```
14.     printf("%f\n", arr[1]);
```

```
15.     fclose(fp); //关闭文件
```

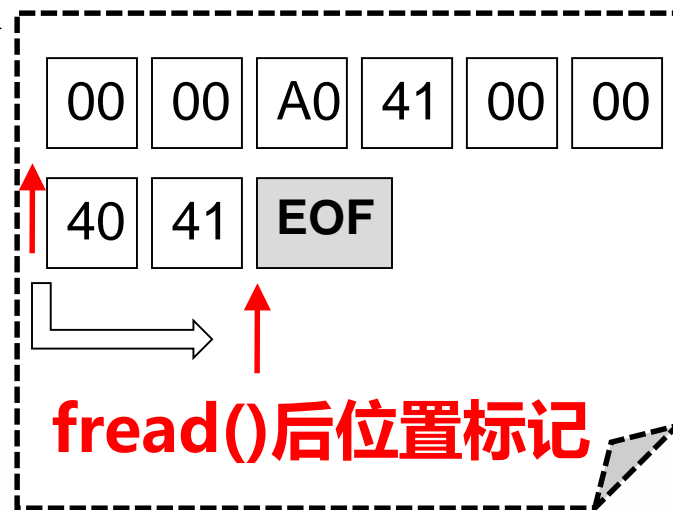
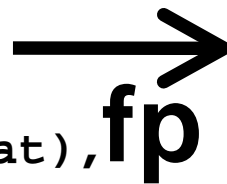
```
16.     return 0;
```

```
17. }
```

```
20.0000000
```

```
12.0000000
```

```
Press any key to continue
```



小结1：四个顺序读文件的函数

- **fgetc**：从指定文本文件读一个字符
- **fgets**：从指定文本文件读一个字符串
- **fscanf**：按格式读取文本文件，可以读取整数，浮点数，字符串等等
- **fread**：从二进制文件读出指定长度的二进制内容

小结2：四个顺序写文件的函数

- **fputc**：向指定文本文件写一个字符
- **fputs**：向指定文本文件写一个字符串
- **fprintf**：按格式写入文本文件，可以写入整数，浮点数，字符串等等
- **fwrite**：向二进制文件写入指定长度的二进制内容

C语言中的流

- 从C程序的角度来看，作为输入输出的各种文件或设备都是统一以**逻辑数据流**的方式出现的。
- C语言把文件看作是一个字符（或字节）的序列。一个输入输出流就是一个字符流或字节（内容为二进制数据）流。

什么是“流”

- “流” (stream)是一种逻辑上的概念
- C语言中，任何输入/输出的数据都视为流
- 输入输出是数据传送的过程，数据如流水一样从一处流向另一处，因此常将输入输出形象地称为流，即**数据流**。流表示了信息从源到目的端的流动。

流是程序与运行环境之间的通道

- 操作系统为程序提供运行环境，无论是用 Word 打开或保存文件，还是 C 程序中的输入输出都是通过操作系统进行的
- 输入操作时，数据从文件流向计算机内存；输出操作时，数据从计算机内存流向文件
- “流”是一个**传输通道**，数据可以从运行环境流入程序中，或从程序流至运行环境

流式文件

➤ C 语言中，数据文件的特点

- ◆ 由一连串的字符（或字节）组成，而不考虑行的界限
- ◆ 两行数据间不会自动加分隔符，对文件的存取是以字符（字节）为单位的，而不考虑记录的界限
- ◆ 输入输出数据流的开始和结束仅受程序控制而不受物理符号（如回车换行符）控制

➤ 这种文件称为流式文件。

三个无需打开的特殊流式文件

- 程序中使用3个标准的流文件：**标准输入流**、**标准输出流**、**标准出错输出流**。
 - ◆ 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(stdin)是从终端的输入
 - 标准输出流(stdout)是向终端的输出
 - 标准出错输出流(stderr)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

向文件读写字符的例子

例：从键盘输入一些字符，逐个把它们送到磁盘上去，直到用户输入一个“#”为止。

➤ **解题思路：**用getchar函数从键盘逐个输入字符，然后用fputc函数写到磁盘文件即可。


```
1. #include <stdio.h>
2. #include <stdlib.h>
3. int main()
4. {
5.     FILE *fp;
6.     char ch, filename[10];
7.     printf("请输入所用的文件名:");
8.     scanf("%s", filename);
9.     if ((fp=fopen(filename, "w"))==NULL)
10.    {
11.        printf("无法打开此文件\n");
12.        exit(0);
13.    }
14.    ch = getchar();
15.    printf("请输入一个字符串(以#结束):");
16.    ch = getchar();
17.    while (ch!='#')
18.    {
19.        fputc(ch, fp);
20.        putchar(ch);
21.        ch=getchar();
22.    }
23.    fclose(fp);
24.    putchar(10);
25.    return 0;
26.}
```

用exit函数时加

输入文件名

以写方式打开文件

接收最后输入的回车符，
原因稍后解释

向打开的文件写入字符

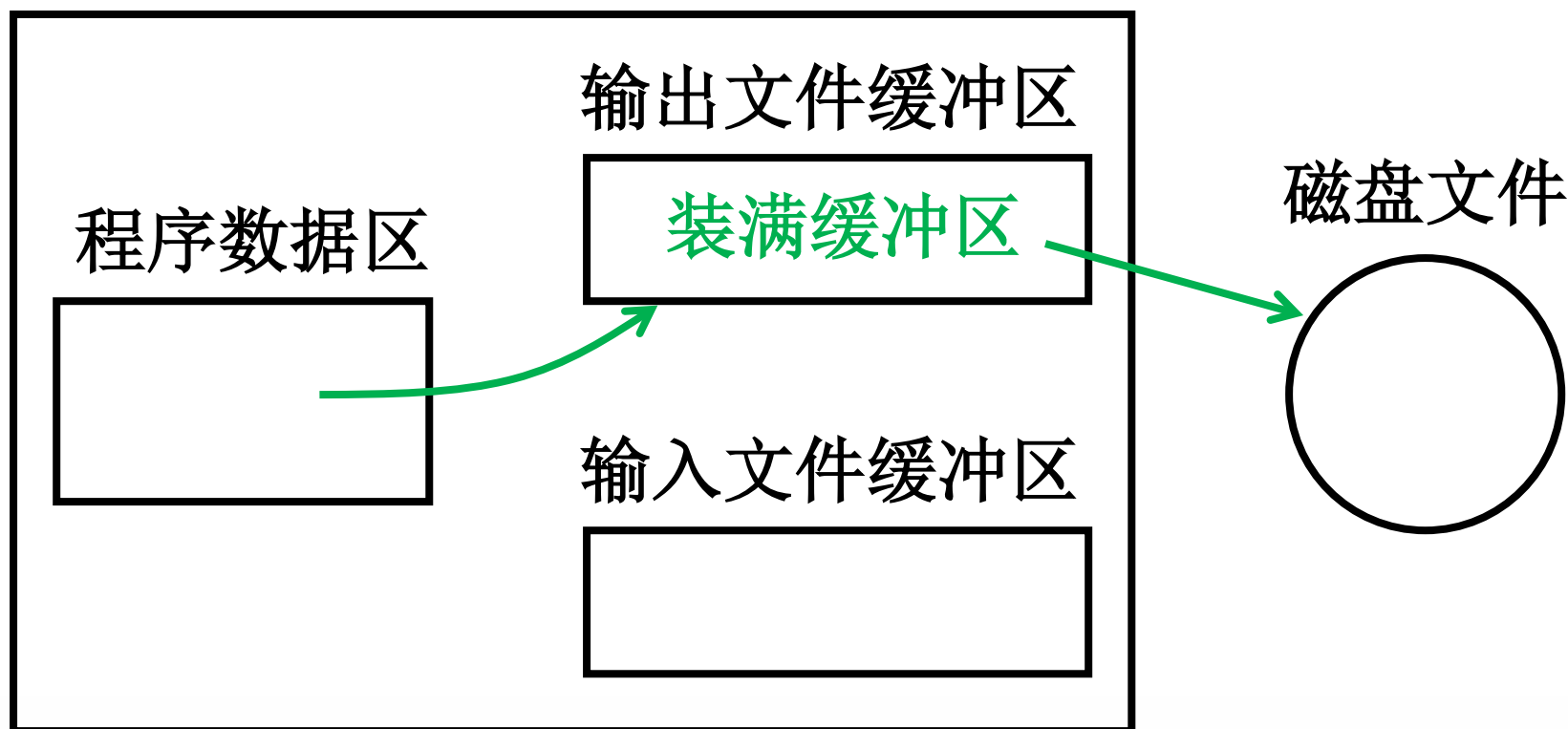
要记得关闭文件，否则前功尽弃！

文件缓冲区

- ANSI C标准采用“缓冲文件系统”处理数据文件
- 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
- 缓存的作用：由于I/O调用很费时间，提供缓存的目的是减少I/O调用的次数

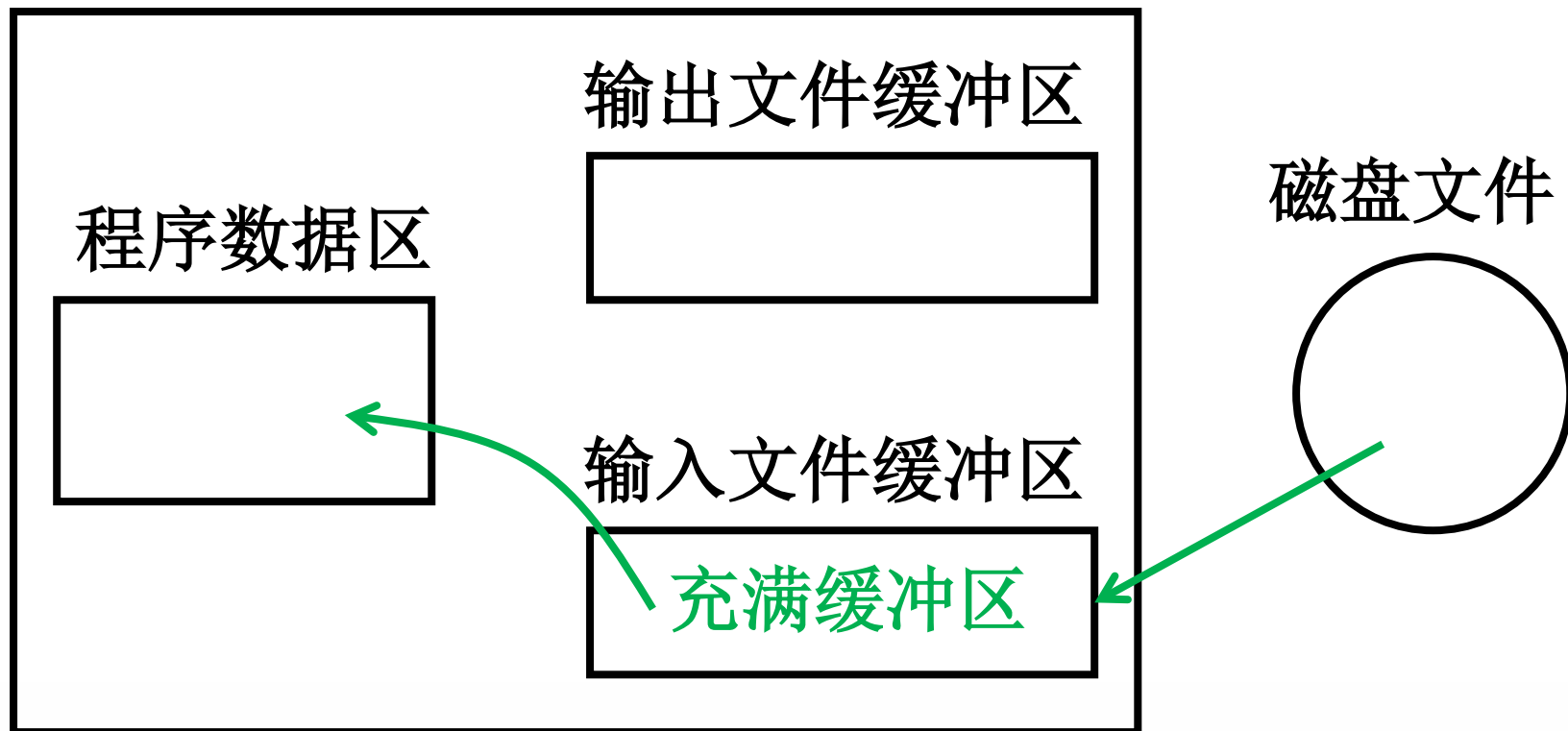
文件缓冲区

- 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



文件缓冲区

- **从磁盘向计算机读入数据**：一次从磁盘文件将一批数据输入到内存缓冲区（充满缓冲区），然后再从缓冲区逐个地将数据送到程序数据区（给程序变量）



缓冲区的具体体现

.....

```
scanf("%s", filename);
```

// 输入的末尾按下了回车键，但是'\n'并没有被filename接收，

// 所以'\n'留在了缓冲区！

```
if ((fp=fopen(filename, "w"))==NULL)
```

```
{
```

```
    printf("无法打开此文件\n");
```

```
    exit(0);
```

```
}
```

```
ch = getchar(); //于是就需要用getchar “吃掉” 那个'\n'
```

.....

结构体声明揭示FILE的真面目

```
typedef struct
{
    short level; //记录打开文件流的缓冲区填入数据的情况
    unsigned flags; //文件状态标志
    char fd; //文件句柄，也就是文件描述符
    unsigned char hold; //若无缓冲区，则不读取字符
    short bsize; //文件缓冲区大小
    unsigned char *buffer; //文件缓冲区指针
    unsigned char *curp; //当前激活的文件位置指针
    unsigned istemp; //临时文件标识
    short token; //用于有效性检查
}FILE;
```

向文件写字符串的例子

例：从键盘读入若干个字符串，对它们按字母大小的顺序排序，然后把排好序的字符串送到磁盘文件中保存。

➤ **解题思路：**为解决问题，可分为三个步骤

- ◆从键盘读入 n 个字符串，存放在一个二维字符数组中，每一个一维数组存放一个字符串；
- ◆对字符数组中的 n 个字符串按字母顺序排序，排好序的字符串仍存放在字符数组中；
- ◆将字符数组中的字符串顺序输出。

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. int main()
5. {
6.     FILE *fp;
7.     char str[3][10],temp[10];
8.     int i,j,k,n=3;
9.     printf("Enter strings:\n");
10.    for (i=0;i<n;i++) //接受三个字符串输入
11.    {
12.        gets(str[i]);
13.    } //未完待续
```


//紧接上页

```
14.  for (i=0;i<n-1;i++) //选择排序
15.  {
16.      k = i;
17.      for (j=i+1;j<n;j++)
18.          if (strcmp(str[k],str[j])>0)
19.              k=j;
20.      if (k!=i)
21.      {
22.          strcpy(temp,str[i]);
23.          strcpy(str[i],str[k]);
24.          strcpy(str[k],temp);
25.      }
26.  } //未完待续
```

//紧接上页

```
27.  if ((fp=fopen("D:\\CC\\string.dat", "w"))==NULL)
28.  {
29.      printf("can't open file!\n");
30.      exit(0);
31.  }
32.  printf("\nThe new sequence:\n");
33.  for (i=0;i<n;i++)
34.  {
35.      fputs(str[i],fp);
36.      fputs("\n",fp);
37.      printf("%s\n",str[i]);
38.  }
39.  fclose(fp);
40.  return 0;
41. }
```

人为地输出一个'\n'

Enter strings:

CHINA

CANADA

INDIA

The new sequence:

CANADA

CHINA

INDIA

从文件中读字符串的例子

➤ 思考：

◆ 从上一个例子的文件string.dat中读回字符串，并在屏幕上显示，应如何编写程序？

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. int main()
4. {
5.     FILE *fp;
6.     char str[3][10];
7.     int i=0;
8.     if ((fp=fopen("D:\\CC\\string.dat","r"))==NULL)
9.     {
10.         printf("can't open file!\n");
11.         exit(0);
12.     }
13.     while (fgets(str[i],10,fp)!=NULL)
14.     {
15.         printf("%s",str[i]);
16.         i++;
17.     }
18.     fclose(fp);
19.     return 0;
20. }
```



CANADA
CHINA
INDIA

不用人为地输出'\n'

使用fgets的注意事项

➤ 调用时可以写成：`fgets(str,n,fp);`

◆ `fgets()` 函数中**第一个参数**可以是静态定义的字符数组，也可以是动态分配的字符数组。作为参数，数组不用写数组名称后面的中括号以及其中的数组长度。

● 例子中**`fgets(str[i],10,fp)`**是怎么回事？因为例子中`str`的定义是一个二维数组！

◆ **第二个参数是“字符串实际长度+1”，因为还有一个‘\0’。**规定的长度应该与第一个参数字符数组的长度相等，若过大，运行程序会有溢出的错误。

◆ `fgets()`和`gets()`不同，当读到‘\n’不停止，仅把‘\n’作为一个字符读入

用二进制方式读写文件举例

例：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。

➤ 解题思路：

- ◆ 定义有10个元素的结构体数组，用来存放10个学生的数据
- ◆ 从main函数输入10个学生的数据
- ◆ 用save函数实现向磁盘输出学生数据
- ◆ 用fwrite函数一次输出一个学生的数据

```
1. #include <stdio.h>
2. #define SIZE 10
3. struct Student_type
4. {
5.     char name[10];
6.     int num;
7.     int age;
8.     char addr[15];
9. } stud[SIZE];
```

```
10. void save()
11. {
12.     FILE *fp;
13.     int i;
14.     if ((fp=fopen("stu.dat","wb"))==NULL)
15.     {
16.         printf("cannot open file\n");
17.         return;
18.     }
19.     for (i=0;i<SIZE;i++)
20.     {
21.         if (fwrite(&stud[i],
22.                     sizeof(struct Student_type),
23.                     1,
24.                     fp) != 1)
25.         {
26.             printf("file write error\n");
27.         }
28.     }
29.     fclose(fp);
30. }
```

当前路径下的文件

10+4+4+15=33，实际上开辟36字节，是4的倍数


```
31.int main()  
32.{  
33.    int i;  
34.    printf("enter data of students:\n");  
35.    for (i=0;i<SIZE;i++)  
36.    {  
37.        scanf("%s%d%d%s",  
38.            stud[i].name, &stud[i].num,  
39.            &stud[i].age, stud[i].addr);  
40.    }  
41.    save();  
42.    return 0;  
43.}
```

用二进制方式读取并显示文件

- 为了验证在磁盘文件 “stu.dat” 中是否已存在此数据，可以用以下程序从 “stu.dat” 文件中读入数据，然后在屏幕上输出。

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #define SIZE 10
4. struct Student_type
5. {
6.     char name[10];
7.     int num;
8.     int age;
9.     char addr[15];
10. } stud[SIZE];
```

```
11.int main( )
12.{
13.    int i;
14.    FILE *fp;
15.    if ((fp=fopen("stu.dat","rb"))==NULL)
16.    {
17.        printf("cannot open file\n");
18.        exit(0);
19.    }
20.    for (i=0;i<SIZE;i++)
21.    {
22.        fread(&stud[i], sizeof(struct Student_type),
23.            1, fp);
24.        printf("%-10s %4d %4d  %-15s\n",
25.            stud[i].name,stud[i].num,
26.            stud[i].age,stud[i].addr);
27.    }
28.    fclose(fp);
29.    return 0;
30.}
```

用二进制方式实现文件复制

- 如果修改前面的例子：从已有的二进制文件“stu.list”中，读入数据并输出到“stu.dat”文件中，应如何修改程序？

- **解题思路：**

- ◆ 编写load函数

- ◆ main函数中先调用load函数，再调用前面提到的save函数

```
1. void save()
2. {
3.     FILE *fp;
4.     int i;
5.     if ((fp=fopen("stu.dat","wb"))==NULL)
6.     {
7.         printf("cannot open file\n");
8.         return;
9.     }
10.    for (i=0;i<SIZE;i++)
11.    {
12.        if (fwrite(&stud[i],
13.                    sizeof(struct Student_type),
14.                    1,
15.                    fp) != 1)
16.        {
17.            printf("file write error\n");
18.        }
19.    }
20.    fclose(fp);
21.}
```

```
1. void load()  
2. {  
3.     FILE *fp;  
4.     int i;  
5.     if ((fp=fopen("stu_list","rb"))==NULL)  
6.     {  
7.         printf("cannot open infile\n");  
8.         return;  
9.     }  
10.    for (i=0;i<SIZE;i++)  
11.        if (fread(&stud[i],sizeof(struct student_type),  
12.                1,fp)!=1)  
13.        {  
14.            if (feof(fp))  
15.            {  
16.                fclose(fp);  
17.                return;  
18.            }  
19.            printf("file read error\n");  
20.        }  
21.    fclose (fp);  
22. }
```

```
int main()  
{  
    load();  
    save();  
    return 0;  
}
```

文本文件与二进制

- 首先，应明确所有文件本质上都是二进制序列，只是文本文件的每个字节分别代表一个字符（ASCII码），所以是个**字符流**
- 程序要使用文本文件中存储的数据计算，就需要把字符流转成**二进制流**，比如需要把字符串“100”转换成整数100。而存成文本文件又需要反过来转换一次。

二进制文件的优缺点

- 二进制文件的**优点**：节约不必要的转换时间
- 二进制文件的**不足**：
 - ◆ 一台计算机或一个操作系统上产生的二进制文件另一计算机或操作系统可能无法识别（因为不像ASCII编码一样有统一的格式）
 - ◆ 人们无法像在文本处理软件中检查和修改文本文件那样来处理二进制文件，因为无法直接理解

文本文件 vs. 二进制文件 (1)

示例编号	处理文本文件	处理二进制文件	作用
1	<pre>file1_in_t= fopen("file1.txt","r"); file2_in_t= fopen("file2.txt", "r");</pre>	<pre>file1_in_t= fopen("file1.bin","rb"); file2_in_t= fopen("file2.bin", "rb");</pre>	打开两个输入文件
2	<pre>file1_out_t= fopen("file3.txt", "w"); file2_out_t= fopen("file4.txt", "w");</pre>	<pre>file1_in_t= fopen("file3.bin","wb"); file2_in_t= fopen("file4.bin", "wb");</pre>	打开两个输出文件

文本文件 vs. 二进制文件 (2)

示例编号	处理文本文件	处理二进制文件	作用
3	<pre>fscanf(file1_in_t, "%s%s%s%d%d", ticket.departure, ticket.destination, ticket.time, &ticket.num, &ticket.fee);</pre>	<pre>fread(&ticket, sizeof(struct trainticket), 1, file1_in_b);</pre>	将列车车票结构从数据文件读取到内存
4	<pre>fprintf(file1_in_t, "%s%s%s%d%d", ticket.departure, ticket.destination, ticket.time, &ticket.num, &ticket.fee);</pre>	<pre>fwrite(&ticket, sizeof(struct trainticket), 1, file1_out_b);</pre>	将列车车票结构写入到输出文件中

文本文件 vs. 二进制文件 (3)

示例编号	处理文本文件	处理二进制文件	作用
5	<pre>for(i=0; i<SIZE;i++) { fscanf(file2_in_t, "%d",&array[i]); }</pre>	<pre>fread(array, sizeof(int), SIZE, file2_in_b);</pre>	用输入文件中的int型数值填充array数组
6	<pre>for(i=0; i<SIZE;i++) { fprintf(file2_out_t, "%d\n",array[i]); }</pre>	<pre>fwrite(array, sizeof(int), SIZE, file2_out_b);</pre>	将数组array的内容写入输出文件

文本文件 vs. 二进制文件 (4)

示例编号	处理文本文件	处理二进制文件	作用
7	<pre>fclose(file1_in_t); fclose(file2_in_t); fclose(file1_out_t); fclose(file2_out_t);</pre>	<pre>fclose(file1_in_b); fclose(file2_in_b); fclose(file1_out_b); fclose(file2_out_b);</pre>	关闭所有输入输出文件

作业 2017/12/29

➤ 按下列要求编写程序，提交手写源代码

1. 声明一个结构体类型，其中包含姓名和生日两个成员（都是字符串）。编程实现从键盘输入3位家人的姓名和生日，调用fwrite函数，将这些信息写入到二进制文件family.dat中。然后调用fread函数，读取文件中的全部记录，并显示到屏幕上。
2. 编程实现两个文本文件内容的逐字符比较。若两文件内容完全相同，则输出“Same”；若两个文件有不同之处，则输出从文件开始处到第一个出现不同的地方为止有多少个字符是相同的。