# 第10章 对文件的输入输出

# 10.1 C文件的有关基本知识

# 10.1.1 什么是文件

- 文件有不同的类型，在程序设计中，主要用到两种文件：

(1) 程序文件。包括源程序文件(后缀为.c)、目标文件(后缀为.obj)、可执行文件(后缀为.exe)等。这种文件的内容是程序代码。

# 10.1.1 什么是文件

- 文件有不同的类型，在程序设计中，主要用到两种文件：

  (2) 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，如在程序运行过程中输出到磁盘(或其他外部设备)的数据，或在程序运行过程中供读入的数据。如一批学生的成绩数据，或货物交易的数据等。

- 本章主要讨论的是数据文件

# 10.1.1 什么是文件

- 在以前各章中所处理的数据的输入和输出，从终端的键盘输入数据，运行结果输出到终端显示器上
- 常常需要将一些数据输出到磁盘上保存起来，以后使用
- 这就要用到磁盘文件

# 10.1.1 什么是文件

- 操作系统把各种设备都统一作为文件处理
- 从操作系统的角度看，每一个与主机相联的输入输出设备都看作是文件。例如，
  - 终端键盘是输入文件
  - 显示屏和打印机是输出文件

# 10.1.1 什么是文件

- “文件”指存储在外部介质上数据的集合
  - 一批数据是以文件的形式存放在外部介质上的
  - 操作系统是以文件为单位对数据进行管理
  - 想找存放在外部介质上的数据，先按文件名找到所指定的文件，然后再从该文件读数据
  - 要向外部介质上存储数据也必须先建立一个文件（以文件名作为标志），才能向它输出数据

# 10.1.1 什么是文件

- 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(stream)，即数据流。流表示了信息从源到目的端的流动。

# 10.1.1 什么是文件

- 输入操作时，数据从文件流向计算机内存
- 输出操作时，数据从计算机流向文件
- 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
- “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

# 10.1.1 什么是文件

- 从C程序的观点来看，无论程序一次读写一个字符，或一行文字，或一个指定的数据区，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。

# 10.1.1 什么是文件

- C 的数据文件由一连串的字符（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。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，这就增加了处理的灵活性。这种文件称为流式文件。

# 10.1.2 文件名

- 文件要有一个唯一的文件标识，以便用户识别和引用。
- 文件标识包括三部分：

  (1)文件路径

  (2)文件名主干

  (3)文件后缀

# 10.1.2 文件名

文件路径 文件名主干 文件后缀

置。

D: \CC\temp\file1.dat

◆表示file1.dat文件存放在D盘中的CC目录下的temp子目录下面

# 10.1.2 文件名

- 文件路径表示文件在外部存储设备中的位置。如：

文件名

**D: \CC\temp\file1.dat**

  - 表示**file1.dat**文件存放在**D**盘中的**CC**目录下的**temp**子目录下面

# 10.1.2 文件名

命名规则遵循标识符的命名规则

- 文件路径表[illegible]设备中的位置。如：

  **D: \CC\temp\file1.dat**

  - 表示**file1.dat**文件存放在**D**盘中的**CC**目录下的**temp**子目录下面

一般不超过3个字母（doc、txt、dat、c、cpp、obj、exe、ppt、bmp等）

- 中的位置。如：

  **D: \CC\temp\file1.dat**

  ◆表示**file1.dat**文件存放在**D**盘中的**CC**目录下的**temp**子目录下面

若有char name[50];

如何用name储存以上文件名？

strcpy(name,”D: \CC\temp\file1.dat”);

strcpy(name,”D: \\CC\\temp\\file1.dat”);

# 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，如果不加转换地输出到外存，就是二进制文件
  - 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，每一个字节放一个字符的**ASCII**代码

# 10.1.3 文件的分类

- 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
  - 如有整数**10000**，如果用**ASCII**码形式输出到磁盘，则在磁盘中占 5 个字节(每一个字符占一个字节)，而用二进制形式输出，则在磁盘上只占**4**个字节(用**VC++ C**时)

# 10.1.3 文件的分类

ASCII形式

| 00110001 | 00110000 | 00110000 | 00110000 | 00110000 |
|---|---|---|---|---|
| （1） | （0） | （0） | （0） | （0） |

二进制形式

| 00000000 | 00000000 | 00100111 | 00010000 |
|---|---|---|---|
| （10000） | | | |

# 10.1.4 文件缓冲区

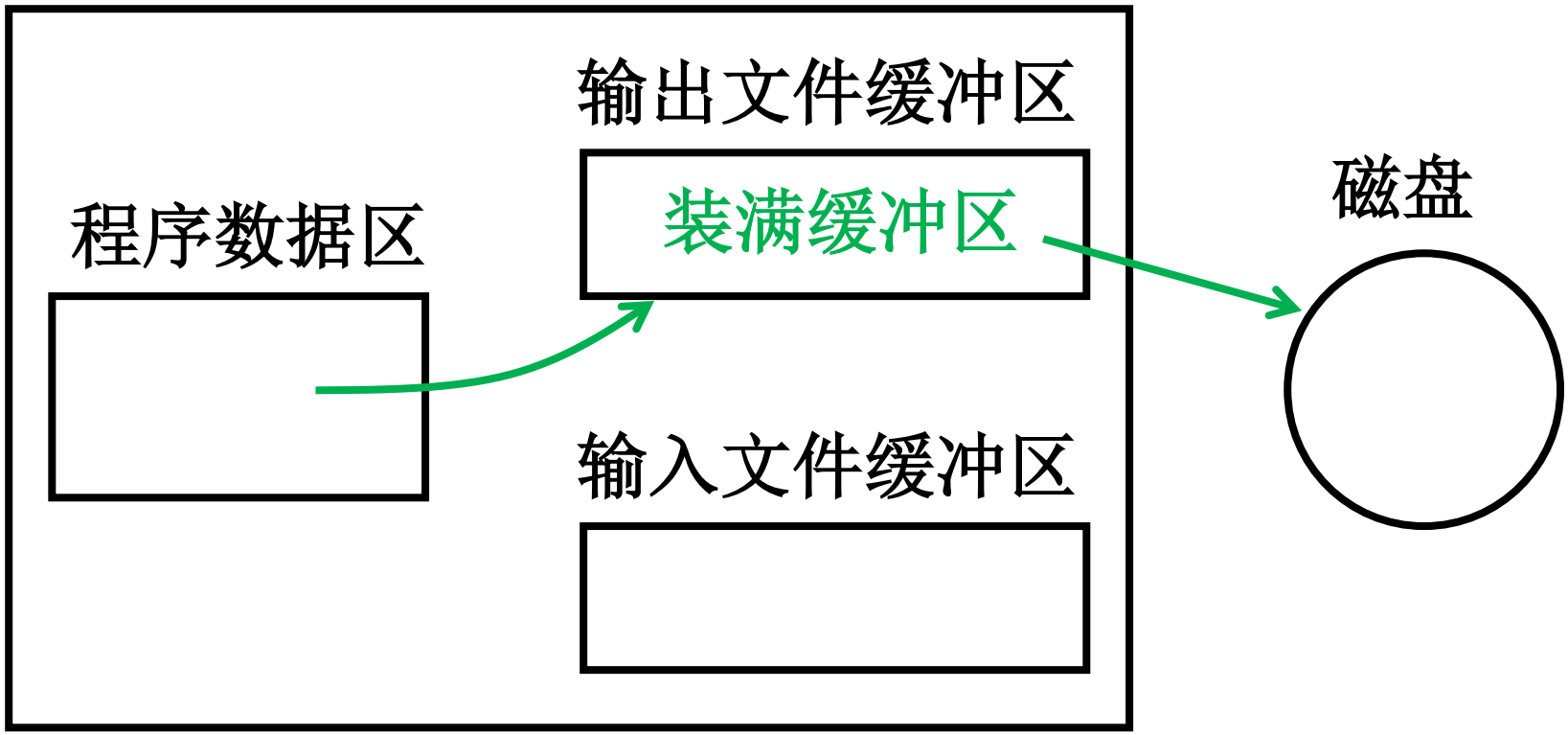
- **ANSI C**标准采用“缓冲文件系统”处理数据文件
- 所谓缓冲文件系统是指系统自动地在内存区为程序中每一个正在使用的文件开辟一个文件缓冲区

# 10.1.4 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（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）

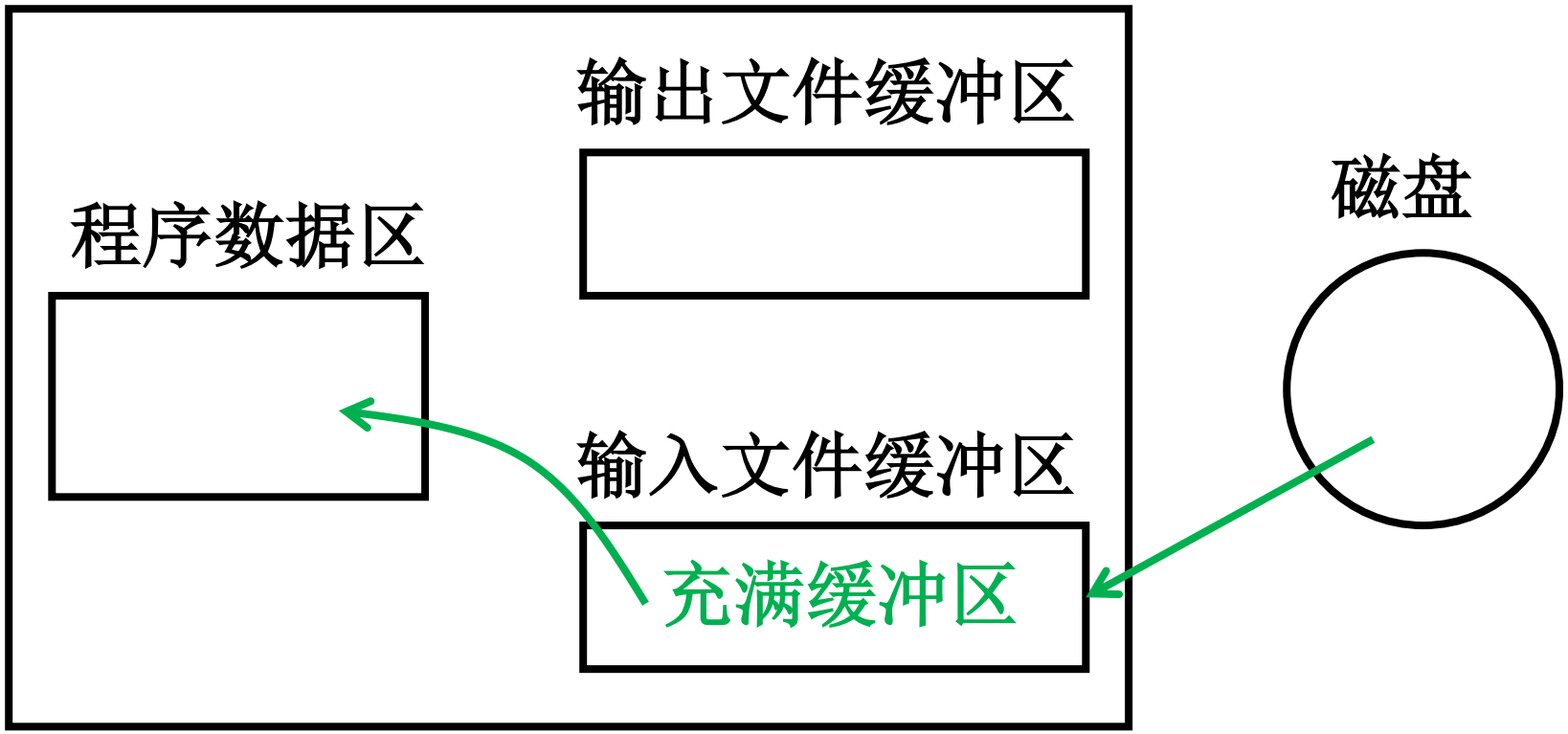
# 10.1.4 文件缓冲区

- 从内存向磁盘输出数据



# 10.1.4 文件缓冲区

- 从磁盘向计算机读入数据



# 10.1.5 文件类型指针

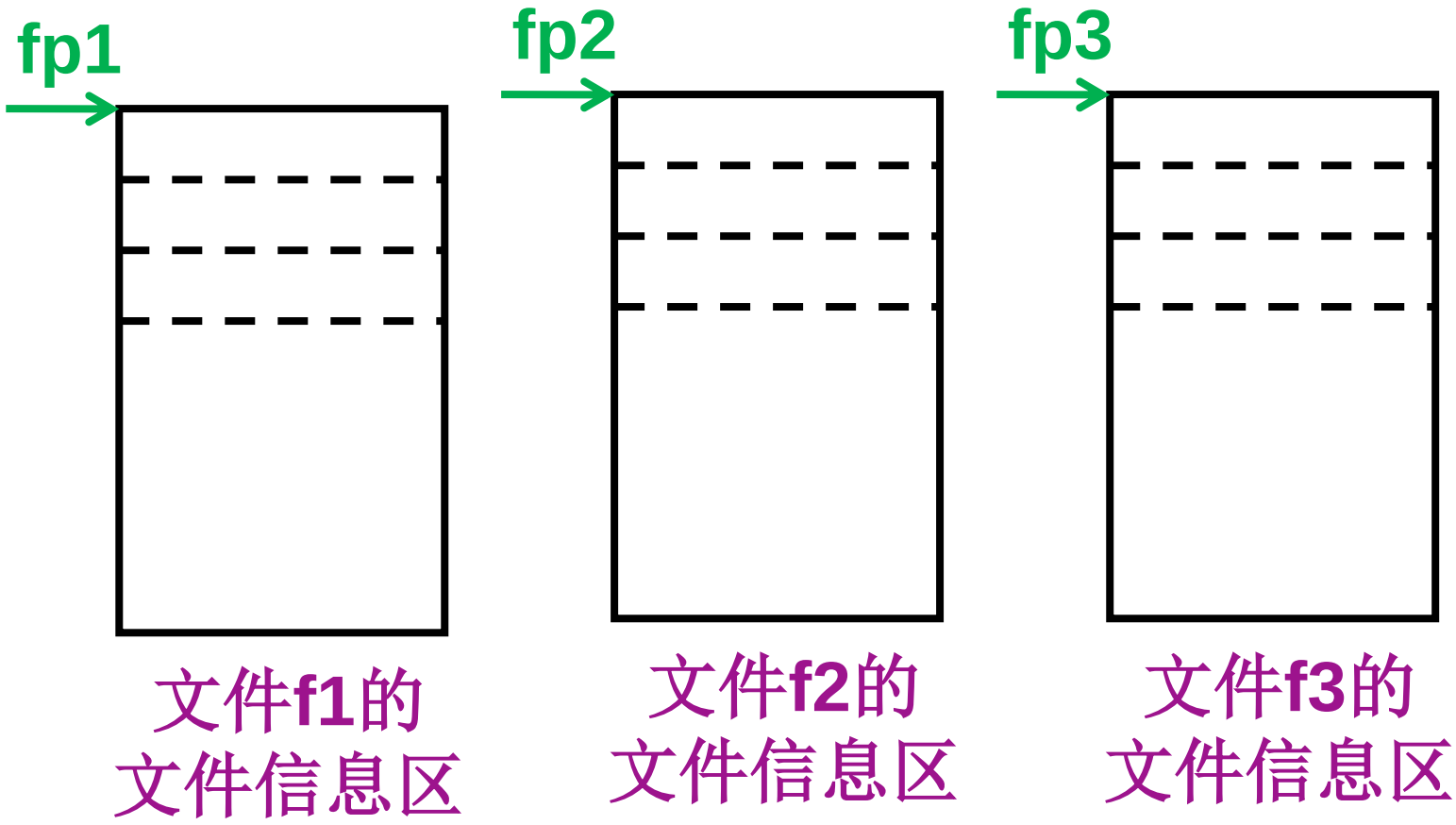
- 缓冲文件系统中，关键的概念是“文件类型指针”，简称“文件指针”
  - 每个被使用的文件都在内存中开辟一个相应的文件信息区，用来存放文件的有关信息（如文件的名字、文件状态及文件当前位置等）
  - 这些信息是保存在一个结构体变量中的。该结构体类型是由系统声明的，取名为**FILE**

# 10.1.5 文件类型指针

- 声明FILE结构体类型的信息包含在头文件“stdio.h”中
- 一般设置一个指向FILE类型变量的指针变量，然后通过它来引用这些FILE类型变量

# 10.1.5 文件类型指针

```
FILE  *fp1,*fp2,*fp3;
```



# 10.2 打开与关闭文件

## 10.2.1 用fopen函数打开数据文件

## 10.2.2 用fclose函数关闭数据文件

# 10.2.1 用fopen函数打开数据文件

- 对文件读写之前应该“打开”该文件，在使用结束之后应“关闭”该文件。
- 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)。

# 10.2.1 用fopen函数打开数据文件

- 在编写程序时，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
- 所谓“关闭”是指撤销文件信息区和文件缓冲区

# 10.2.1 用fopen函数打开数据文件

- fopen函数的调用方式为：

  fopen(文件名,使用文件方式);

- 例如：

  fopen("a1","r");

  - 表示要打开名为“a1”的文件，使用文件方式为“读入”
  - fopen函数的返回值是指向a1文件的指针

# 10.2.1 用fopen函数打开数据文件

- 通常将**fopen**函数的返回值赋给一个指向文件的指针变量。如：

```
FILE *fp;
fp=fopen("a1","r");
```

  - ◆**fp**和文件**a1**相联系，**fp**指向了**a1**文件

# 10.2.1 用fopen函数打开数据文件

- 在打开一个文件时，通知编译系统以下3个信息：

  ①需要访问的文件的名字

  ②使用文件的方式（“读”还是“写”等）

  ③让哪一个指针变量指向被打开的文件

改错题

- 使用文件方式参见教材表10.1。

- 说明：

  (1) 用“r”方式打开的文件只能用于向计算机输入而不能用作向该文件输出数据，而且该文件应该已经存在，并存有数据，这样程序才能从文件中读数据。

  ◆不能用“r”方式打开一个并不存在的文件，否则出错。

➢说明：

(2) 用“w”方式打开的文件只能用于向该文件写数据（即输出文件），而不能用来向计算机输入。

◆如果原来不存在该文件，则在打开文件前新建立一个以指定的名字命名的文件。

◆如果原来已存在一个以该文件名命名的文件，则在打开文件前先将该文件删去，然后重新建立一个新文件。

- 说明：

**(3)** 如果希望向文件末尾添加新的数据（不希望删除原有数据），则应该用“**a**”方式打开

◆但此时应保证该文件已存在；否则将得到出错信息。

◆打开文件时，文件读写标记移到文件末尾

➢ 说明：

(4) 用r+、w+、a+方式打开的文件既可以用来输入数据，也可以用来输出数据。

◆用r+方式时该文件应该已经存在。

◆用w+方式则新建立一个文件，先向此文件写数据，然后可以读此文件中的数据。

◆用a+方式打开的文件，原来的文件不被删去，文件读写位置标记移到文件末尾，可以添加，也可以读。

- 说明：

  **(5)** 如果打开失败，**fopen**函数将会带回一个出错信息。**fopen**函数将带回一个空指针值**NULL**

- 常用下面的方法打开一个文件：

```
if ((fp=fopen("file1", " r"))==NULL)
 {printf("cannot open this file\n");
  exit(0);
 }
```

文件打开的常用语句

终止正在执行的程序

- 说明：

  (6) C标准建议用表10.1列出的文件使用方式打开文本文件或二进制文件，但目前使用的有些C编译系统可能不完全提供所有这些功能

- 说明：

(7) 计算机输从**ASCII**文件读入字符时，遇到回车换行符，系统把它转换为一个换行符，在输出时把换行符转换成为回车和换行两个字符。在用二进制文件时，不进行这种转换，在内存中的数据形式与输出到外部文件中的数据形式完全一致，一一对应。

- 说明：

  (8) 程序中可以使用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。所以以前我们用到的从终端输入或输出到终端都不需要打开终端文件。

# 10.2.2 用fclose函数关闭数据文件

- 关闭文件用**fclose**函数。**fclose**函数调用的一般形式为

  **fclose(**文件指针**);**

  例如：

  **fclose （fp);**

- 如果不关闭文件将会丢失数据。

↑

改错！！

# 10.3 顺序读写数据文件

- 在顺序写时，先写入的数据存放在文件中前面，后写入的数据存放在文件中后面
- 在顺序读时，先读文件中前面的数据，后读文件中后面的数据
- 对顺序读写来说，对文件读写数据的顺序和数据在文件中的物理顺序是一致的
- 顺序读写需要用库函数实现

# 10.3 顺序读写数据文件

# 10.3.1 怎样向文件读写字符

- 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 | 写成功，返回值就是输出的字符；输出失败，则返回ＥＯＦ（即-1） |

例**10.1** 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**fgetc**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。

```
#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( );
```

用exit函数时加

输入文件名

只写

接收最后输入的回车符

```
    printf("请输入一个字符串(以#结束)：");
    ch=getchar( );
    while(ch!='#')
    {  fputc(ch,fp);
       putchar(ch);
       ch=getchar();
     }
    fclose(fp);
    putchar(10);
    return 0;
}
```

不要遗漏，改错！

例**10.2** 将一个磁盘文件中的信息复制到另一个磁盘文件中。 今要求将上例建立的**file1.dat**文件中的内容复制到另一个磁盘文件**file2.dat**中。

- 解题思路：处理此问题的算法是：从**file1.dat**文件中逐个读入字符，然后逐个输出到**file2.dat**中。

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{ FILE *in,*out;
  char  ch,infile[10],outfile[10];
  printf("输入读入文件的名字:");
  scanf("%s",infile);
  printf("输入输出文件的名字:");
  scanf("%s",outfile);
  if((in=fopen(infile,"r"))==NULL)
  {                                  文件\n"); exit(0);}
  i                                  file,"w"))==NULL)
  {                                  文件\n"); exit(0); }
```

改为rb和wb，则复制一个二进制文件

```
ch=fgetc(in);
while(!feof(in))
 {fputc(ch,out);
   putchar(ch);
   ch=fgetc(in);
   }
   putchar(10);
   fclose(in);
   fclose(out);
   return 0;
}
```

检查当前读写位置
是否移到文件末尾

# 10.3.2 怎样向文件读写一个字符串

- 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) | str所指向的字符串写到文件指针变量fp所指向的文件中 | 写成功，返回0；否则返回非0值 |

- 说明：

**fgets函数的函数原型为：**

**char *fgets (char *str,int n,FILE *fp);**

- ◆其作用是从文件读入一个字符串
- ◆调用时可以写成：

**fgets(str,n,fp);**

➢说明：

注意：改错

◆**fgets(str,n,fp);**中**n**是要求得到的字符个数，但实际上只读**n-1**个字符，然后在最后加一个**'\0'**字符，这样得到的字符串共有**n**个字符，把它们放到字符数组**str**中

◆如果在读完**n-1**个字符之前遇到换行符“**\n**”或文件结束符**EOF**，读入即结束，但将所遇到的换行符“**\n**”也作为一个字符读入

◆执行**fgets**成功，返回**str**数组首地址，如果一开始就遇到文件尾或读数据错，返回**NULL**

➢说明：

**fputs函数的函数原型为：**

**int fputs (char *str, FILE *fp);**

◆**str**指向的字符串输出到**fp**所指向的文件中

◆调用时可以写成： **fputs(”China”,fp);**

◆**fputs**函数中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

◆输出成功，函数值为0；失败，函数值为**EOF**

例**10.3** 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。

```
#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("Enter strings:\n");
  for(i=0;i<n;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("D:\\CC\\string.dat",
                              "w"))==NULL)
    {printf("can't open file!\n"); exit(0);}
    printf("\nThe new sequence:\n");
    for(i=0;i<n;i++)
    { fputs(str[i],fp);
      fputs("\n",fp);
      printf("%s\n",str[i]);
    }  fclose(fp);
  return 0;
}
```

人为地输出一个'\n'

- 思考：
  - 从文件**string.dat**中读回字符串，并在屏幕上显示，应如何编写程序？

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ FILE *fp;  char  str[3][10];  int i=0;
   if((fp=fopen("D:\\CC\\string.dat",
   "r"))==NULL)
   {printf("can't open file!\n");exit(0);}
   while(fgets(str[i],10,fp)!=NULL)
   { printf("%s",str[i]);  i++;  }
  fclose (fp);
  return 0;
}
```

不用人为地输出'\n'

# 10.3.3用格式化的方式读写文件

- 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);**

需要进行格式转化，速度较慢，尽量不用

# 10.3.4 用二进制方式向文件读写一组数据

- 一般调用形式为:

**fread(buffer，size，count，fp);**

**fwrite(buffer，size，count，fp);**

# 10.3.4 用二进制方式向文件读写一组数据

地址，注意取地址符号&不要遗漏，改错！

- **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

例10.4 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

```
#include <stdio.h>
#define SIZE 10
struct Student_type
{ char name[10];
   int num;
   int age;
   char addr[15];
 }stud[SIZE];
```

当前路径下的文件

```
void save( )
{ FILE *fp;   int i;
  if((fp=fopen("stu.dat","wb"))==NULL)
  { printf("cannot open file\n");
    return;
  }
  for(i=0;i<SIZE;i++)
    if(fwrite(&stud[i],
     sizeof(struct Student_type),
  1,fp)!=1)
       printf("file write error\n");
  fclose(fp);
}
```

```
int main()
{ int i;
   printf("enter data of students:\n");
   for(i=0;i<SIZE;i++)
     scanf("%s%d%d%s",
   stud[i].name,&stud[i].num,
  &stud[i].age,stud[i].addr);
   save( );
   return 0;
}
```

- 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。

```
#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( )
{int i;   FILE *fp;
 if((fp=fopen("stu.dat","rb"))==NULL)
 {printf("cannot open file\n"); exit(0); }
 for(i=0;i<SIZE;i++)
 {fread
  (&stud[i],sizeof(structStudent_type),1,fp);
   printf ("%-10s %4d %4d  %-15s\n",
              stud[i].name,stud[i].num,
                 stud[i]. age,stud[i].addr);
  }
  fclose (fp);
  return 0;
}
```

- 如果修改例**10.4**：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**函数中再调用**load**函数

```
void load( )
{ FILE *fp;   int i;
  if((fp=fopen("stu_list","rb"))==NULL)
  {printf("cannot open infile\n"); return;}
  for(i=0;i<SIZE;i++)
    if(fread(&stud[i],sizeof(struct
        student_type),1,fp)!=1)
    { if(feof(fp))
        {   fclose(fp);   return;    }
      printf("file read error\n");
    }
  fclose (fp);
}
```

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

# 10.4 随机读写数据文件

- 对文件进行顺序读写比较容易理解，也容易操作，但有时效率不高
- 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 随机读写数据文件

## 10.4.1 文件位置标记及其定位

## 10.4.2 随机读写

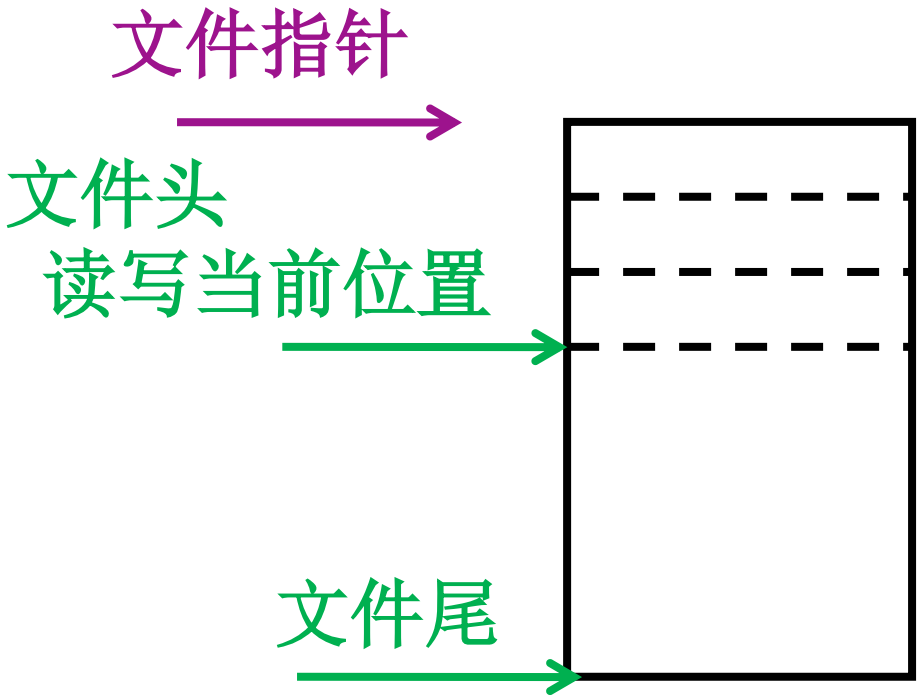
# 10.4.1 文件位置标记及其定位

## 1.文件位置标记

- 为了对读写进行控制，系统为每个文件设置了一个文件读写位置标记(简称文件标记)，用来指示“接下来要读写的下一个字符的位置”

# 10.4.1 文件位置标记及其定位

## 1.文件位置标记



# 10.4.1 文件位置标记及其定位

## 1.文件位置标记

- 一般情况下，在对字符文件进行顺序读写时，文件标记指向文件开头，进行读的操作时，就读第一个字符，然后文件标记向后移一个位置，在下一次读操作时，就将位置标记指向的第二个字符读入。依此类推，直到遇文件尾，结束

# 10.4.1 文件位置标记及其定位

## 1.文件位置标记

- 如果是顺序写文件，则每写完一个数据后，文件标记顺序向后移一个位置，然后在下一次执行写操作时把数据写入指针所指的位置。直到把全部数据写完，此时文件位置标记在最后一个数据之后

# 10.4.1 文件位置标记及其定位

## 1.文件位置标记

- 可以根据读写的需要，人为地移动了文件标记的位置。文件标记可以向前移、向后移，移到文件头或文件尾，然后对该位置进行读写——随机读写
- 随机读写可以在任何位置写入数据，在任何位置读取数据

# 10.4.1 文件位置标记及其定位

## 2. 文件位置标记的定位

◆可以强制使文件位置标记指向指定的位置

◆可以用以下函数实现：

**(1)用rewind函数使文件标记指向文件开头**

**rewind**函数的作用是使文件标记重新返回文件的开头，此函数没有返回值。

例**10.5** 有一个磁盘文件，内有一些信息。要求第一次将它的内容显示在屏幕上，第二次把它复制到另一文件上。

- 解题思路：
  - 因为在第一次读入完文件内容后，文件标记已指到文件的末尾，如果再接着读数据，就遇到文件结束标志，**feof**函数的值等于**1**(真)，无法再读数据
  - 必须在程序中用**rewind**函数使位置指针返回文件的开头

```
#include<stdio.h>
int main()
{ FILE *fp1,*fp2;
  fp1=fopen("file1.dat","r");
  fp2=fopen("file2.dat","w");
  ch=getc(fp1);
  while(!feof(fp1))
  {     putchar(ch);
        ch=getc(fp1);
   }
  putchar(10);
  rewind(fp1);
  ch=getc(fp1);
  while(!feof(fp1))
   {    fputc(ch,fp2);
        ch=fgetc(fp1);
   }
  fclose(fp1);   fclose(fp2);
  return 0;
}
```

为什么需要这个语句？

# 10.4.1 文件位置标记及其定位

## 2. 文件位置标记的定位

◆可以强制使文件标记指向指定的位置

◆可以用以下函数实现：

**(2) 用fseek函数改变文件标记**

**fseek函数的调用形式为：**

**fseek(文件类型指针,位移量,起始点)**

◆起始点0代表“文件开始位置”，1为“当前位置”，2为“文件末尾位置”

- C 标准指定的名字

| 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。位移量应是**long**型数据**(**在数字的末尾加一个字母**L)**。
- **fseek**函数一般用于二进制文件。下面是**fseek**函数调用的几个例子：
  - **fseek (fp,100L,0);**
  - **fseek (fp,50L,1);**
  - **fseek (fp,-10L,2);**

# 10.4.1 文件位置标记及其定位

## 2. 文件位置标记的定位

◆可以强制使文件位置标记指向指定的位置

◆可以用以下函数实现：

**(3) 用ftell函数测定文件位置标记的当前位置**

**ftell**函数的作用是得到流式文件中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**函数得到当前位置，用相对于文件开头的位移量来表示。如果调用函数时出错（如不存在**fp**指向的文件），**ftell**函数返回值为**-1L**。例如：

```
i=ftell(fp);
if(i==-1L) printf("error\n");
```

文件开始位置的“文件位置”标记为**0**

# 10.4.2 随机读写

例10.6 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。

- 要求：从例10.4中建立的“stu.dat”中读入数据

# 10.4.2 随机读写

- 解题思路：
  - 按二进制只读方式打开文件
  - 将文件位置标记指向文件的开头，读入一个学生的信息，并把它显示在屏幕上
  - 再将文件标记指向文件中第3，5，7，9个学生的数据区的开头，读入相应学生的信息，并把它显示在屏幕上
  - 关闭文件

```
#include<stdio.h>
#include <stdlib.h>
struct St
{ char name[10];
   int num;
   int age;
   char addr[15];
}stud[10];
```

```
int main()
{ int i;  FILE *fp;
  if((fp=fopen("stu.dat","rb"))==NULL)
  { printf("can not open file\n"); exit(0); }
   for(i=0;i<10;i+=2)
   { fseek(fp,i*sizeof(struct St),0);
     fread(&stud[i], sizeof(struct St),1,fp);
     printf("%-10s %4d %4d %-15s\n",
         stud[i].name,stud[i].num,
         stud[i].age,stud[i].addr);
  }
  fclose(fp);  return 0;
}
```

# 10.5 文件读写的出错检测

## 1.ferror函数

- ferror函数的一般调用形式为

  ferror(fp);

  - ◆如果返回值为0，表示未出错，否则表示出错
  - ◆每次调用输入输出函数，都产生新的ferror函数值，因此调用输入输出函数后立即检查
  - ◆调用fopen时，ferror的初始值自动置为0

# 10.5 文件读写的出错检测

## 2. clearerr函数

- 作用是使文件错误标志和文件结束标志置为0
- 调用一个输入输出函数时出现错误（ferror值为非零值），立即调用clearerr(fp)，使ferror(fp)值变0，以便再进行下一次检测
- 只要出现文件读写错误标志，它就一直保留，直到对同一文件调用clearerr函数或rewind函数，或任何其他一个输入输出函数

```
#include <stdio.h>
int main()
{       FILE *fp; int e; char c; long len;
        fp=fopen("a.tmp","w");
        fputs("1B345678", fp);
        fclose(fp);
        fp=fopen("a.tmp","r");
        len=ftell(fp);
        printf("%ld\n",len);
        fscanf(fp,"%d%c",&e,&c);
        len=ftell(fp);
        printf("%ld\n",len);
        fseek(fp, -1L, SEEK_END);
        len=ftell(fp);
        printf("%ld\n",len);
        return 0;
}
```

```
0
2
7
```

```
#include <stdio.h>
int main()
{     FILE *fp; int e; char c; long len;
      fp=fopen("a.tmp","w");
      fputs("1B345D78", fp);
      fclose(fp);
      fp=fopen("a.tmp","r");
      fseek(fp,3L,0);
      fscanf(fp,"%d%c",&e,&c);
      printf("%d,%c\n",e,c);
      return 0;
}
```

45,D

```
FILE *fp;
double d=2015.108;
char c1,c2;
int i;
char str[4];
fp=fopen("a.tmp","w");
fprintf(fp,"%.4f", d);
fclose(fp);
fp=fopen("a.tmp","r");
c1=fgetc(fp);
fgets(str, 4, fp);
c2=fgetc(fp);
fscanf(fp, "%d", &i);
printf("%c,%s,%c,%d",c1,
      str,c2,i);
fclose(fp);
```

**2,015,.,1080**

```
FILE *fp; char str[8]; int n;
fp=fopen("a.tmp","w");
fputs("goodbye 2019", fp);
fclose(fp);
fp=fopen("a.tmp","r");
fgets(str, 3, fp);
fseek(fp, -3L, SEEK_END);
fscanf(fp, "%d", &n);
printf("%s,%d\n", str, n);
```

go,19