第8章　输入、输出和文件系统

在前面的程序设计中，我们介绍了输入和输出，即从标准输入设备 —— 键盘输入，由标准输出设备 —— 显示器或打印机输出。不仅如此，我们也常把磁盘作为信息载体，用于保存中间结果或最终数据。在使用一些字处理工具时，会利用打开一个文件来将磁盘的信息输入到内存，通过关闭一个文件来实现将内存数据输出到磁盘。这时的输入和输出是针对文件系统，故文件系统也是输入和输出的对象，谈到输入和输出，自然也离不开文件系统。

文件可以从不同的角度来分类：

1) 按文件所依附的介质来分：有卡片文件、纸带文件、磁带文件、磁盘文件等。

2) 按文件内容来分：有源文件、目标文件、数据文件等。

3) 按文件中数据组织形式分：有字符文件和二进制文件。

字符文件通常又称为ASCII码文件或正文文件，按字符存储，具有可读性；而二进制文件是以二进制存储，不具备可读性，但从存储空间的利用来看，实型数无论位数大小均占 4位，字符确需按位数来存放，这样的话，二进制文件相对就节省了空间。

目前C语言使用的文件系统分为缓冲文件系统（标准I/O）和非缓冲文件系统（系统I/O）。

8.1　缓冲文件系统

缓冲文件系统的特点是：在内存开辟一个“缓冲区”，为程序中的每一个文件使用，当执行读文件的操作时，从磁盘文件将数据先读入内存“缓冲区”，装满后再从内存“缓冲区”依此读入接收的变量。执行写文件的操作时，先将数据写入内存“缓冲区”，待内存“缓冲区”装满后再写入文件。由此可以看出，内存“缓冲区”的大小，影响着实际操作外存的次数，内存“缓冲区”越大，则操作外存的次数就少，执行速度就快、效率高。一般来说，文件“缓冲区”的大小随机器而定。

8.1.1　文件的打开与关闭

任何关于文件的操作都要先打开文件，再对文件进行读写，操作完毕后，要关闭文件。

1. 文件类型指针

人们在操作文件时，通常都关心文件的属性，如文件的名字、文件的性质、文件的当前状态等。对缓冲文件系统来说，上述特性都是要仔细考虑的。 ANSI C为每个被使用的文件在内存开辟一块用于存放上述信息的小区，利用一个结构体类型的变量存放。该变量的结构体类型由系统取名为FILE，在头文件stdio.h中定义如下：

```
typedef struct{
 int_fd;          /*文件号*/
 int_cleft;       /*缓冲区中的剩余字符*/
 int_mode;        /*文件的操作模式*/
 char*_next;      /*下一个字符的位置*/
 char *_buff;     /*文件缓冲区的位置*/
```

```
} FILE;
```

在操作文件以前，应先定义文件变量指针：

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

按照上面的定义，fp1和fp2均为指向结构体类型的指针变量，分别指向一个可操作的文件，换句话说，一个文件有一个文件变量指针，今后对文件的访问，会转化为针对文件变量指针的操作。

2. 文件的打开

ANSI C 提供了打开文件的函数：

```
FILE  *fopen(char *fname,char *mode)
```

函数原型在stdio.h文件中，fopen()打开一个fname指向的外部文件，返回与它相连接的流。fname是字符串，应是一个合法的文件名，还可以指明文件路经。对文件的操作模式由 mode 决定，mode也是字符串，由表8-1给出mode的取值表。

表8-1 mode的取值表

Mode	含 义
r	打开一个文本文件只读
w	打开一个文本文件只写
a	打开一个文本文件在尾部追加
rb	打开一个只读的二进制文件
wb	打开一个只写的二进制文件
ab	对二进制文件追加
r+	打开一个可读/写的文本文件
w+	创建一个新的可读/写的文本文件
a+	打开一个可读/写的文本文件
rb+	打开一个可读/写的二进制文件
wb+	创建一个新的可读/写的二进制文件
ab	打开一个可读/写的二进制文件

如表8-1所示，文件的操作方式有文本文件和二进制文件两种，打开文件的正确方法如下例所示：

```
#include <stdio.h>
FILE  *fp;

If ((fp=fopen("test.txt","w"))==NULL)
     {  /*创建一个只写的新文本文件*/
          printf("cannot open file \n");
          exit(0);
     }
```

这种方法能发现打开文件时的错误。在开始写文件之前检查诸如文件是否有写保护，磁盘是否已写满等，因为函数会返回一个空指针 NULL，NULL值在stdio.h中定义为0。事实上打开文件是要向编译系统说明三个信息：①需要访问的外部文件是哪一个。②打开文件后要执行读或写即选择操作方式。③确定哪一个文件指针指向该文件。对打开文件所选择的操作方式来说，一经说明不能改变，除非关闭文件后重新打开。是只读就不能对其写操作，对已存

文件如以新文件方式打开，则信息必丢失。

3. 文件的关闭

ANSI C 提供了关闭文件的函数：

```
int fclose(FILE *stream)
```

fclose()函数关闭与stream相连接的文件，并把它的缓冲区内容全部写出。在 fclose()函数调用以后，流stream与此文件无关，同时原自动分配的缓冲区也失去定位。

fclose()函数关闭文件操作成功后，函数返回 0；失败则返回非零值。

[例8-1] 打开和关闭一个可读可写的二进制文件：

```
#include <stdio.h>

   main()
  {
   FILE *fp;
  If ((fp=fopen("test.dat","rb"))==NULL)
   {
      printf("cannot open file\n");
   exit(0);
   }
/*写入对文件执行读写的代码
……              */
if (fclose(fp)) printf("file close error!\n");
}
```

8.1.2 文件的读写

当文件按指定的工作方式打开以后，就可以执行对文件的读和写。下面按文件的性质分类进行操作。针对文本文件和二进制文件的不同性质，对文本文件来说，可按字符读写或按字符串读写；对二进制文件来说，可进行成块的读写或格式化的读写。

1. 读写字符

C提供fgetc和fputc函数对文本文件进行字符的读写，其函数的原型存于 stdio.h头文件中，格式为：

```
int fgetc(FILE *stream)
```

fgetc()函数从输入流的当前位置返回一个字符，并将文件指针指示器移到下一个字符处，如果已到文件尾，函数返回 EOF，此时表示本次操作结束，若读写文件完成，则应关闭文件。

```
int fputc(int ch,FILE *stream)
```

fputc（）函数完成将字符 ch的值写入所指定的流文件的当前位置处，并将文件指针后移一位。fputc ()函数的返回值是所写入字符的值，出错时返回 EOF。

[例8-2] 将存放于磁盘的指定文本文件按读写字符方式逐个地从文件读出，然后再将其显示到屏幕上。采用带参数的 main()，指定的磁盘文件名由命令行方式通过键盘给定。

```
#include <stdio.h>
main( argc,argv)
int argc;
```

```
char *argv[];
{
 char ch;
 FILE *fp;
 int i;
 if((fp=fopen(argv[1],"r"))==NULL)          /* 打开一个由argv[1] 所指的文件 */
  {
   printf("not open");
   exit(0);
   }
 while ((ch=fgetc(fp))!=EOF)                /* 从文件读一字符，显示到屏幕 */
      putchar(ch);
 fclose(fp);
}
```

程序是一带参数的main()函数，要求以命令行方式运行，其参数 argc是用于记录输入参数的个数，argv是指针数组，用于存放输入参数的字符串，串的个数由 argc描述。假设我们指定读取的文件名为 L8-2.c，并且列表文件内容就是源程序。经过编译和连接生成可执行的文件 L8-2.exe。运行程序l8-2.exe，输入的命令行方式为：

```
c:\tc> l8-2 L8-2.c
```

上述程序以命令行方式运行，其输入参数字符串有两个，即 argv[0]="c:\tc>l8-2"、argv[1]=" L8-2.c "，argc=2。故打开的文件是L8-2.c 。程序中对fgetc()函数的返回值不断进行测试，若读到文件尾部或读文件出错，都将返回 C的整型常量EOF，其值为非零有效整数。程序的运行输出为源程序本身：

```
c:\tc> l8-2 L8-2.c
```

```
#include <stdio.h>
main( argc,argv)
int argc;
char *argv[];
{
 char ch;
 FILE *fp;
 int i;
 if((fp=fopen(argv[1],"r"))==NULL)          /* 打开一个由argv[1] 所指的文件 */
  {
   printf("not open");
   exit(0);
   }
   while ((ch=fgetc(fp))!=EOF)              /* 从文件读一字符，显示到屏幕 */
   putchar(ch);
   fclose(fp);
}
```

[例8-3] 从键盘输入字符，存到磁盘文件 test.txt中：

```
#include <stdio.h>
main()
```

```
{
 FILE fp;                                  /* 定义文件变量指针 */
 char ch;
 if((fp=fopen("test.txt","w"))==NULL)      /* 以只写方式打开文件*/
{
  printf("cannot open file!\n");
 exit(0);
  }
while ((ch=fgetchar())!='\n')              /*只要输入字符非回车符*/
 fputc(ch,fp)                              /*写入文件一个字符 */
 fclose(fp);
}
```

程序通过从键盘输入一以回车结束的字符串，写入指定的流文件 test.txt，文件以文本只写方式打开，所以流文件具有可读性，能支持各种字符处理工具访问。简单地说，我们可以通过DOS提供的type命令来列表显示文件内容。

运行程序：

```
RUN
I love china!
```

在DOS操作系统环境下，利用 type 命令显示test.txt文件如下：

```
c:\tc> type test.txt
I love china!
```

2. 读写字符串

C提供读写字符串的函数原型在 stdio.h头文件中，其函数形式为：

```
Char *fgets(char *str,int num,FILE *stream)
```

fgets() 函数从流文件stream中读取至多 num-1个字符，并把它们放入str指向的字符数组中。读取字符直到遇见回车符或 EOF（文件结束符）为止，或读入了所限定的字符数。

```
 int fputs(char *str,FILE *stream)
```

fputs()函数将str指向的字符串写入流文件。操作成功时，函数返回 0值，失败返回非零值。

[例8-4] 向磁盘写入字符串，并写入文本文件 test.txt：

```
#include <stdio.h>
#include <string.h>
main()
{
 FILE *fp;
 char str[128];
 if ((fp=fopen("test.txt","w"))==NULL打开只写的文本文件*/
   {
   printf("cannot open file!");
   exit(0);
   }
  while((strlen(gets(str)))!=0)
     {  /*若串长度为零，则结束*/
      fputs(str,fp);                       /*写入串*/
      fputs("\n",fp);   /写入回车符*/
```

```
    }
        fclose(fp); /*关文件*/
}
```

运行该程序，从键盘输入长度不超过127个字符的字符串，写入文件。如串长为0，即空串，程序结束。

输入：Hello!

```
How do you do
Good-bye!

_
```

运行结束后，我们利用dos的type命令列表文件：

```
c:\tc> type test.txt
   Hello!
   How do you do
   Good-bye!
```

这里所输入的空串，实际为一单独的回车符，其原因是gets函数判断串的结束是以回车作标志的。

[例8-5] 从一个文本文件test1.txt中读出字符串，再写入令一个文件test2.txt。

```
#include <stdio.h>
#include <string.h>
main()
{
 FILE *fp1,*fp2;
 char str[128];
 if ((fp1=fopen("test1.txt","r"))==NULL)
    {     /*以只读方式打开文件1*/
     printf("cannot open file\n");
     exit(0);
     }
if ((fp2=fopen("test2.txt","w"))==NULL)
    {  /*以只写方式打开文件2*/
     printf("cannot open file\n");
     exit(0);
     }
 while ((strlen(fgets(str,128,fp1)))>0)
      /*从文件中读回的字符串长度大于0 */
     {
         fputs(str,fp2 );      /*从文件1读字符串并写入文件2*/
         printf("%s",str);     /*在屏幕显示*/
     }
    fclose(fp1);
fclose(fp2);
}
```

程序共操作两个文件，需定义两个文件变量指针，因此在操作文件以前，应将两个文件以需要的工作方式同时打开（不分先后），读写完成后，再关闭文件。设计过程是按写入文件

的同时显示在屏幕上，故程序运行结束后，应看到增加了与原文件相同的文本文件并显示文件内容在屏幕上。

3. 格式化的读写

前面的程序设计中，我们介绍过利用 scanf()和printf()函数从键盘格式化输入及在显示器上进行格式化输出。对文件的格式化读写就是在上述函数的前面加一个字母 f成为fscanf()和fprintf()。其函数调用方式：

```
int fscanf(FILE *stream,char *format,arg_list)
int fprintf(FILE *stream,char *format,arg_list)
```

其中，stream为流文件指针，其余两个参数与 scanf()和printf()用法完全相同。

[例8-6] 将一些格式化的数据写入文本文件，再从该文件中以格式化方法读出显示到屏幕上，其格式化数据是两个学生记录，包括姓名、学号、两科成绩。

```
#include <stdio.h>
main()
{
   FILE *fp;
   int i;
   struct stu{          /*定义结构体类型*/
   char name[15];
   char num[6];
   float score[2];
   }student;            /*说明结构体变量*/
   if ((fp=fopen("test1.txt","w"))==NULL)
    {                   /*以文本只写方式打开文件*/
     printf("cannot open file");
     exit(0);
     }
  printf("input data:\n");
  for( i=0;i<2;i++)
     {
     scanf("%s %s %f %f",student.name,student.num,&student.score[0],
     &student.score[1]);      /*从键盘输入*/
     fprintf(fp,"%s %s %7.2f %7.2f\n",student.name,student.num,
     student.score[0],student.score[1]);/*      写入文件*/
     }
     fclose(fp);          /*关闭文件*/

  if ((fp=fopen("test.txt","r"))==NULL)
     { /*以文本只读方式重新打开文件*/
     printf("cannot open file");
     exit(0);
     }
  printf("output from file:\n");
  while (fscanf(fp,"%s %s %f %f\n",student.name,student.num,
  &student.score[0],student.score[1])!=EOF       )
  /*从文件读入*/
     printf("%s %s %7.2f %7.2f\n",student.name,student.num,
```

```
        student.score[0],student.score[1]);显示到屏幕*/
    fclose(fp); /*关闭文件*/
}
```

程序设计一个文件变量指针，两次以不同方式打开同一文件，写入和读出格式化数据，有一点很重要，那就是用什么格式写入文件，就一定用什么格式从文件读，否则，读出的数据与格式控制符不一致，就造成数据出错。上述程序运行如下：

```
input data:
xiaowan j001 87.5 98.4
xiaoli j002 99.5 89.6
output from file:
xiaowan j001   87.50   98.40
xiaoli j002   99.50   89.60
```

列表文件的内容显示为：

```
c:\> type test.txt
   xiaowan j001   87.50   98.40
   xiaoli  j002   99.50   89.60
```

此程序所访问的文件也可以定为二进制文件，若打开文件的方式为：

```
if ((fp=fopen("test1.txt","wb"))==NULL)
 {              /*以二进制只写方式打开文件*/
  printf("cannot open file");
  exit(0);
  }
```

其效果完全相同。

4. 成块读写

前面介绍的几种读写文件的方法，对其复杂的数据类型无法以整体形式向文件写入或从文件读出。C语言提供成块的读写方式来操作文件，使其数组或结构体等类型可以进行一次性读写。成块读写文件函数的调用形式为：

```
int fread(void *buf,int size,int count,FILE *stream)
int fwrite(void *buf,int size,int count,FILE *stream)
```

fread（）函数从stream指向的流文件读取count（字段数）个字段，每个字段为size(字段长度)个字符长，并把它们放到buf（缓冲区）指向的字符数组中。

fread（）函数返回实际已读取的字段数。若函数调用时要求读取的字段数超过文件存放的字段数，则出错或已到文件尾，实际在操作时应注意检测。

fwrite()函数从buf(缓冲区)指向的字符数组中，把count(字段数)个字段写到stream所指向的流中，每个字段为size个字符长，函数操作成功时返回所写字段数。

关于成块的文件读写，在创建文件时只能以二进制文件格式创建。

[例8-7] 向磁盘写入格式化数据，再从该文件读出显示到屏幕。

```
#include "stdio.h"
#include "stdlib.h"
main()
{
 FILE *fp1;
```

```
    int i;
    struct stu{     /*定义结构体*/
            char name[15];
            char num[6];
            float score[2];
            }student;
    if ((fp1=fopen("test.txt","wb"))==NULL)
            { /*以二进制只写方式打开文件*/
            printf("cannot open file");
            exit(0);
            }
    printf("input data:\n");
     for( i=0;i<2;i++) {
                scanf("%s %s %f %f",student.name,student.num,
              &student.score[0],&student.score[1]);/*       输入一记录*/
           fwrite(&student,sizeof(student),1,fp1);成块写入文件*/
              }
          fclose(fp1);

    if ((fp1=fopen("test.txt","rb"))==NULL)
    { /*重新以二进制只写打开文件*/
    printf("cannot open file");
     exit(0);
     }
   printf("output from file:\n");
   for (i=0;i<2;i++)
             {
             fread(&student,sizeof(student),1,fp1);/*       从文件成块读*/
             printf("%s %s %7.2f %7.2f\n",student.name,student.num,
             student.score[0],student.score[1]);/*       显示到屏幕*/
             }
   fclose(fp1);
     }
```

运行程序：

```
input data:
xiaowan j001 87.5 98.4
xiaoli j002 99.5 89.6
output from file:
xiaowan j001   87.50   98.40
xiaoli j002   99.50   89.60
```

通常，对于输入数据的格式较为复杂的话，我们可采取将各种格式的数据当做字符串输入，然后将字符串转换为所需的格式。 C**提供函数**:

```
int    atoi(char *ptr)
float  atof(char *ptr)
long int atol(char *ptr)
```

它们分别将字符串转换为整型、实型和长整型。使用时请将其包含的头文件 math.h**或**

stdlib.h写在程序的前面。

[例8-8] 将输入的不同格式数据以字符串输入，然后将其转换进行文件的成块读写。

```
#include <stdio.h>
#include <stdlib.h>
main()
{
 FILE *fp1;
 char *temp;
 int i;
 struct stu{                    /*定义结构体类型*/
   char name[15];               /*姓名*/
   char num[6];                 /*学号*/
   float score[2];              /*二科成绩*/
   }student;
 if ((fp1=fopen("test.txt","wb"))==NULL)  /*打开文件*/
    {
    printf("cannot open file");
    exit(0);
    }
for( i=0;i<2;i++) {
    printf("input name:");
    gets(student.name);          /*输入姓名*/
    printf("input num:");
    gets(student.num);           /*输入学号*/
    printf("input score1:");
     gets(temp);                 /*输入成绩*/
     student.score[0]=atof(temp);
     printf("input score2:");
     gets(temp);
     student.score[1]=atof(temp);

     fwrite(&student,sizeof(student),1,fp1);        /*成块写入到文件*/
}
fclose(fp1);

 if ((fp1=fopen("test.txt","rb"))==NULL)
    {
     printf("cannot open file");
     exit(0);
     }
printf("--------------------\n");
printf("%-15s%-7s%-7s%-7s\n","name","num","score1","score2");
printf("--------------------\n");
for (i=0;i<2;i++)
    {
    fread(&student,sizeof(student),1,fp1);
    printf("%-15s%-7s%7.2f%7.2f\n",student.name,student.num,
    student.score[0],student.score[1]);
```

```
    }
    fclose(fp1);
}
```

运行程序如下：RUN

```
input name:li-ying
input num: j0123
input score1 98.65
input score2 89.6
input name:li-li
input num: j0124
input score1 68.65
input score2 86.6
---------------------------------
name         num    score1    score2
---------------------------------
li-ying      j0123   98.65     89.60
li-li        j124    68.64     86.60
```

8.1.3 随机读写文件

随机对文件的读写是指在文件内部任意对文件内容进行访问，这也就需要对文件进行详细的定位，只有定位准确，才有可能对文件随机访问。

C语言提供了用于文件定位的函数，它的作用是使文件指针移动到所需要的位置。

```
int fseek(FILE *fp,long d,int pos)
```

fp是文件指针，d是位移量，pos是起始点。

Pos的取值为：

0 ：文件开始处

1 ：文件的当前位置

2 ：文件的尾部

位移量d是long型的数据，可以为正或负值。表示从起始点向下或向上的指针移动。函数的返回值若操作成功为0，操作失败为非零。

例如：fseek(fp,5L,0)；将文件指针从文件头向下移动5个字节。

fseek(fp,-10L,2)；将文件指针从当前位置向上移动10个字节。

rewind() 将文件指针移动到文件头。

ftell(FILE *fp) 返回文件指针的当前位置。

[例8-9] 写入5个学生记录，记录内容为学生姓名、学号、两科成绩。写入成功后，随机读取第三条记录，并用第二条记录替换。

```
#include <stdio.h>
#include <stdlib.h>
#define n 5
main()
{
 FILE *fp1;      /*定义文件指针*/
```

```
char *temp;
int i,j;
struct stu{            /*定义学生记录结构*/
  char name[15];
  char num[6];
  float score[2];
  }student[n];
if ((fp1=fopen("test.txt","wb"))==NULL)  /*以二进制只写方式打开文件*/
  {
  printf("cannot open file");
  exit(0);
  }
 for( i=0;i<n;i++)
  {

    printf("input name:");   /*输入姓名*/
    gets(student[i].name);
    printf("input num:");
    gets(student[i].num);                    /*输入学号*/
    printf("input score1:");
    gets(temp);                        /*输入一科成绩*/
    student[i].score[0]=atof(temp);
    printf("input score2:");
    gets(temp);                        /*输入第二科成绩*/
    student[i].score[1]=atof(temp);

    fwrite(&student[i],sizeof(struct stu),1,fp1)成块写入*/
    }
  fclose(fp1); /*关闭*/

if ((fp1=fopen("test.txt","rb+"))==NULL)
{   /*以可读写方式打开文件*/
 printf("cannot open file");
 exit(0);
 }
 printf("--------------------\n");
 printf("%-15s%-7s%-7s%-7s\n","name","num","score1","score2");
 printf("--------------------\n");
 for (i=0;i<n;i++)
     {  /*显示全部文件内容*/
    fread(&student[i],sizeof(struct stu),1,fp1);
     printf("%-15s%-7s%7.2f%7.2f\n",student[i].name,student[i].num,
  student[i].score[0],student[i].score[1]);
    }
/*以下进行文件的随机读写*/
  fseek(fp1,3*sizeof(struct stu),0);/*定位文件指针指向第三条记录*/
  fwrite(&student[1],sizeof(struct stu),1,fp1);
         /*在第三条记录处写入第二条记录*/
  rewind(fp1); /*移动文件指针到文件头*/
 printf("--------------------\n");
```

```
    printf("%-15s%-7s%-7s%-7s\n","name","num","score1","score2");
    printf("--------------------\n");

     for (i=0;i<n;i++)
        {  /*重新输出文件内容*/
       fread(&student[i],sizeof(struct stu),1,fp1);
        printf("%-15s%-7s%7.2f%7.2f\n",student[i].name,student[i].num,
     student[i].score[0],student[i].score[1]);
       }

    fclose(fp1);     /*关闭文件*/
  }
```

运行程序：

```
RUN
input name:li-ying
input num: j0123
input score1 98.65
input score2 89.6
input name:li-li
input num: j0124
input score1 68.65
input score2 86.6
input name:li-ping
input num: j0125
input score1 88.5
input score2 84.6
input name:Wang-xian
input num: j0126
input score1 98
input score2:94
input name:Ma-ling
input num: j0127
input score1 66.5
input score2 80.6
---------------------------------
name         num            score1    score2
---------------------------------
li-ying            j0123   98.65    89.60
li-li       j0124   68.64    86.60
li-ping            j0125   88.50    84.60
Wang-xian    j0126  98.0094.00
Ma-ling      j0127 6.50    80.60
---------------------------------
name         num            score1    score2
---------------------------------
li-ying            j0123   98.65    89.60
li-li       j0124   68.64    86.60
li-li       j0124   68.64    86.60
Wang-xian    j0126  98.0094.00
```

```
Ma-ling     j0127    66.50     80.60
```

程序的第二次输出，即随机访问后，文件中会有两条相同的记录。

8.2 非缓冲文件系统

前面介绍的缓冲文件系统是借助文件结构体指针来对文件进行管理，通过文件指针来对文件进行访问，既可以读写字符、字符串、格式化数据，也可以读写二进制数据。非缓冲文件系统依赖于操作系统，通过操作系统的功能对文件进行读写，是系统级的输入输出，它不设文件结构体指针，只能读写二进制文件，但效率高、速度快，由于 ANSI标准不再包括非缓冲文件系统，因此建议大家最好不要选择它。本书只作简单介绍。

1. 文件的打开与关闭

非缓冲文件系统不是 ANSI标准定义的，是 UNIX型I/O系统的一员，所以，其原型位于io.h文件中。

打开文件：

```
int open(char *fname,int access);
```

打开文件名为fname，以access方式访问：

access的值为：O_RDONLY　　只读
　　O_WRONLY　　只写
　　O_RDWR　　读写

关闭文件：

```
close(int fd);
```

下述程序用UNIX系统打开和关闭一个文件：

```
#include "io.h"
#include "fcntl.h"
#include "sys\stat.h"
main(argc,argv)
int argc;
char *argv[]
{
 int fd;
if((fd=open(argv[1],O_RDONLY))==-1) 以只读方式打开文件*/
   {
    printf("cannt open file!");
   exit(0);
   }
printf("file existent!");
if (close(fd)) printf("error in closing file\n");
}
```

2. 文件的读写

对非缓冲文件系统的读写函数的原型在 io.h头文件中，其调用形式为：

```
int read(int fd,void *buf,int count)
```

read()函数从fd说明的文件中读取 count个字节到buf所指向的缓冲区。函数的返回值是实

际读写的字节数。

```
int write(int fd,void *buf,int count)
```

write()函数把count个字节从buf写入到fd说明的文件中。函数的返回值是实际写入的字节数。

下面例子从文件TEST.TST中读取它的前半100个字节并放到数组buffer中。

```
#include "io.h"
#include "stdio.h"
#include "fcntl.h"

main()
{
 int fd;
char buffer[100];
if ((fd=open("TEST.TST",O_RDONLY))==-1)打开文件*/
   {
    printf("cannot open file !\n");
   exit(0);
   }
if (read(fd,buffer,100)!=100)       /*判断读写的字节数是否正确*/
printf("Possible read error.");
}
```

8.3 文件系统应用举例

文件操作在程序设计中是非常重要的技术，文件的数据格式不同，决定了对文件操作方式的不同。

[例8-10] 我们需要同时处理三个文件。文件addr.txt记录了某些人的姓名和地址；文件tel.txt记录了顺序不同的上述人的姓名与电话号码。希望通过对比两个文件，将同一人的姓名、地址和电话号码记录到第三个文件addrtel.txt。首先看一下前两个文件的内容：

```
type addr.txt
hejie       tianjing
liying      shanghai
liming      chengdu
wangpin     chongqing
type tel.txt
liying      12345
hejie       8764
wangpin      87643
liming      7654322
```

这两个文件格式基本一致，姓名字段占14个字符，家庭住址或电话号码长度不超过14个字符，并以回车结束。文件结束的最后一行只有回车符，也可以说是长度为0的串。在两个文件中，由于存放的是同一批人的资料，则文件的记录数是相等的，但存放顺序不同。我们可以任一文件记录为基准，在另一文件中顺序查找相同姓名的记录，若找到，则合并记录存入

第三个文件，将查找文件的指针移到文件头，以备下一次顺序查找。

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
main()
{
  FILE *fptr1,*fptr2,*fptr3;                /*定义文件指针*/
  char temp[15],temp1[15],temp2[15];
  if ((fptr1=fopen("addr.txt","r"))==NULL)/*打开文件*/
     {
       printf("cannot open file");
       exit(0);
     }
  if ((fptr2=fopen("tel.txt","r"))==NULL)
     {
       printf("cannot open file");
       exit(0);
     }
 if ((fptr3=fopen("addrtel.txt","w"))==NULL)
     {
       printf("cannot open file");
       exit(0);
     }
  clrscr();                 /*清屏幕*/
 while(strlen(fgets(temp1,15,fptr1))>1) 读回的姓名字段长度大于1*/
     {
       fgets(temp2,15,fptr1);       /*读地址*/
       fputs(temp1,fptr3);          /*写入姓名到合并文件*/
       fputs(temp2,fptr3);          /*写入地址到合并文件*/
       strcpy(temp,temp1);          /*保存姓名字段*/
       do   /*查找姓名相同的记录*/
     {
       fgets(temp1,15,fptr2);
       fgets(temp2,15,fptr2);
       } while (strcmp(temp,temp1)!=0);
  rewind(fptr2);            /*将文件指针移到文件头，以备下次查找*/
  fputs(temp2,fptr3);       /*将电话号码写入合并文件*/

  }
  fclose(fptr1);            /*关闭文件*/
  fclose(fptr2);
  fclose(fptr3);

  }
```

程序运行后，我们来看一下合并后的文件 addrtel.txt**的内容：**

```
type addrtel.txt
hejie          tianjing
8764
liying         shanghai
12345
liming           chengdu
7654322
wangpin        chongqing
87643
```