第9章　文件I/O

本章讨论C语言的文件系统。正如在第8章中解释的，C++支持两种I/O系统：从C语言中继承来的I/O系统和由C++定义的面向对象的系统。本章讨论C的文件系统（C++的文件系统在第二部分描述）。虽然大多数新代码都使用C++文件系统，由于前一章所给的理由，了解C的文件系统仍然是非常重要的。

9.1　C与C++的文件I/O

有时，关于C文件系统和C++之间的关系会有一些混淆。首先，C++支持全部标准C文件系统。所以，如果将来要将C语言代码转换为C++代码，则不必改变所有的I/O程序。第二，C++也定义了自己的面向对象的I/O系统，包括I/O函数和I/O运算符。C++的I/O系统完全包括了C语言的I/O系统功能并把C文件系统看做是冗余的。虽然通常你可能想要使用C++ I/O系统，但如果喜欢，也可自由地使用C文件系统。当然，大多数C++程序员选择使用C++的I/O系统，其原因参见本书的第二部分。

9.2　流和文件

在开始讨论C语言的文件系统之前，了解流和文件这两个概念之间的差别是很重要的。C的I/O系统给程序员提供了一个独立于物理设备进行信息访问的友好界面。即，C的I/O系统在程序员和实际设备之间提供了一级抽象，这个抽象称为流，而物理设备称为文件。了解流和文件如何相互作用是很重要的。

注意：流和文件的概念对本书第二部分讨论的C++ I/O系统也是非常重要的。

9.3　流

C文件系统可在包括终端、磁盘驱动器和磁带驱动器的众多设备上工作。不管设备有多大差异，文件系统都把它们转换成称为“流”的逻辑设备，且所有流的行为是类似的。因为流具有极大的设备无关性，所以向一个磁盘文件进行写操作的同一函数也可以完成向另一种设备（如控制台）的写操作。有两种类型的流：文本流和二进制流。

9.3.1　文本流

文本流由一系列字符组成，标准C允许（但不要求）一个文本流被组织成多行，以换行字符终止一行。然而，在最后一行上换行字符是可选的（实际上，大多数C/C++编译器都不需要使用换行字符终止文本流）。在文本流中，特定字符的转换是由主机环境要求的，例如，新行可以被转换成回车/换行符对。因而，被写（或读）的字符与存储在外设中的字符之间并无一一对应的关系。同样，由于可能的转换，被写（或读）字符的数量可能与外设中存储的字符不一致。

9.3.2 二进制流

二进制流是指一系列字节，这一系列字节与外设中存储的字节有一对一的关系，也就是说，不存在字符转换。此外，写（或读）字节的数量与外设中存储的字节数相同。然而，根据实现决定的空字节数可以追加到一个二进制流中。例如，这些空字节有时用来补充信息以填满磁盘的某个扇区。

9.4 文件

在C/C++中，文件可以是从磁盘文件到终端或打印机的任何东西。流通过执行打开操作与某个文件联系起来。文件一旦打开，里面的信息就可以在程序与该文件之间交换。

并非所有的文件都有相同的功能。例如，磁盘文件支持随机访问，但某些打印机却不行。这就说明了C语言中I/O系统的一个重要特征：所有的流都是相同的，但文件却不同。

如果文件支持定位请求，那么打开文件也使文件位置指针初始化到文件的开始处。当字符从某个文件读出或写到某文件中时，文件位置指针加1，从而确保可以遍历整个文件。

文件通过关闭操作与特定流断开连接。当关闭一个为输出而打开的文件时，与它相关的流的内容（如果存在）被写到外设中。这一过程通常称为“清空”流，它保证了在磁盘缓冲区中不留下任何信息。当程序正常终止时所有文件都自动关闭，其关闭方式是要么通过返回到操作系统的main()，要么通过调用exit()。如果程序因故障崩溃或调用abort()而终止，则文件不关闭。

与文件相关的流都有一个类型为FILE的文件控制结构。永远不要对它进行任何修改操作。

对编程新手来说，把流和文件分开是没有必要的，只要记住保持接口的一致性这个原则即可。只需要考虑流的作用范围并使用一个文件系统来完成所有的I/O操作。I/O系统自动将原始的输入或输出转换成容易管理的流。

9.5 文件系统基础

C文件系统由几个相互联系的函数构成，最常见的函数列于表9.1中。这些函数要求头文件stdio.h。C++程序也可以使用C++风格的头文件<cstdio>。

头文件stdio.h和<cstdio>提供了I/O函数的原型并定义了三个类型size_t，fpos_t和FILE。size_t与fpos_t一样，都是无符号整数的变种。FILE类型在下一节讨论。

stdio.h和<cstdio>中也定义了几个宏命令。与本章相关的宏命令有：NULL，EOF，FOPEN_MAX，SEEK_SET，SEEK_CUR和SEEK_END。NULL定义一个空指针。宏EOF通常被定义为-1，并且是当输入函数读到文件末尾时返回的值。FOPEN_MAX定义了一个整型值，指定在任一时刻可以打开的文件数。其他几个宏和fseek()一起使用，fseek()是实现文件随机访问的函数。

表9.1 常用的C文件系统函数

名字	功能
fopen()	打开一个文件
fclose()	关闭一个文件
putc()	把一个字符写到文件中
fputc()	同putc()

（续表）

名字	功能
getc( )	从某一文件读一个字符
fgetc( )	同getc( )
fgets( )	从某一文件读一个字符串
fputs( )	把一个字符串写到文件中
fseek( )	在文件中查找一个特定字节
ftell( )	返回当前文件的位置
fprintf( )	输出到磁盘文件上
fscanf( )	从磁盘文件中读数据
feof( )	若到文件尾，返回真值
ferror( )	若出错，返回真值
rewind( )	把文件位置指针重新置于文件的起始位置
remove( )	清除一个文件
fflush( )	清空一个文件

9.5.1 文件指针

文件指针是一个将C I/O系统联系在一起的常见的线程。一个文件指针是一个指向FILE类型结构的指针，它指向定义文件各种属性（如文件名、文件状态及文件当前位置）的信息。本质上，文件指针标识特定的文件，并被相关流使用来指导I/O函数的操作。为了读或写文件，用户程序必须使用文件指针，利用如下语句便可获得一个文件指针变量。

```
FILE *fp;
```

9.5.2 打开一个文件

函数fopen( )打开要使用的流并把它与一个文件相连，然后它返回与那个文件相关联的文件指针。通常，在本书的讨论中，文件指磁盘文件。fopen( )函数的原型如下：

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

其中，filename是指向字符串的指针，它必须是有效文件名，可以包括路径说明。mode指向的串决定了打开文件的方式。mode的合法值列于表9.2中，串“r+b”也可表示为“rb+”。

表9.2 mode的合法值

方式	意义
r	为读操作打开一个文本文件
w	为写操作创建一个文本文件
a	附加到文本文件
rb	为读操作打开一个二进制文件
wb	为写操作创建一个二进制文件
ab	附加到二进制文件
r+	为读/写操作打开一个文本文件
w+	为读/写操作创建一个文本文件
a+	为读/写操作附加或创建一个文本文件
r+b	为读/写操作打开一个二进制文件
w+b	为读/写操作创建一个二进制文件
a+b	为读/写操作附加一个二进制文件

如上所述，fopen( )函数返回一个文件指针，你的程序永远也不能改变该指针的值。如果打开文件失败，则fopen( )返回空指针。

下面的程序用fopen( )打开一个名为TEST的文件用做输出：

```
FILE *fp;
fp = fopen("test", "w");
```

虽然从技术上讲前面的代码是正确的，然而，通常写成这样：

```
FILE *fp;

if ((fp = fopen("test","w"))==NULL) {
  printf("Cannot open file.\n");
  exit(1);
}
```

在程序尝试写到它之前，可用上述方法测试打开过程中的任何错误，如写保护或磁盘满错误。通常，人们希望在进行其他文件操作前确认fopen( )正确完成。

尽管大多数文件的方式是自解释的，但是也需要有一些注释。当打开一个文件进行只读操作时，如果那个文件不存在，那么fopen( )失败。当使用追加方式打开一个文件时，如果那个文件不存在，就会创建它。还有，当打开一个文件进行追加操作时，被写到那个文件的所有新数据将被写到那个文件的末尾处。最初的内容将保持不变。如果当为进行写操作打开一个文件时，这个文件不存在，那么就创建它。如果它存在，那么以前存在的文件的内容被删除并且创建一个新文件。在方式r+和w+之间的区别是：如果不存在文件，r+将不会创建一个文件，然而，w+则会。此外，如果文件已经存在，用w+方式打开它会删除它的内容，用r+方式打开它则不会。

正如表9.2所示，文件既能以文本方式打开，也能以二进制方式打开。在大多数实现中，在文本方式下，输入时回车/换行序列转换为换行符；在输出时则相反，换行符转换为回车/换行对。在二进制方式中，不发生这种转换。

在任一时刻可以打开的文件数由FOPEN_MAX指定。这个值通常为8，但请参阅编译手册确认此值。

9.5.3 关闭一个文件

函数fclose( )关闭一个由fopen( )打开的流文件，函数fclose( )把留在磁盘缓冲区中的数据写入文件并在操作系统级正式关闭文件。关闭流文件失败会产生各种麻烦，如丢失数据、破坏文件和程序中出现间歇的错误等。fclose( )也释放与流相关联的文件控制块，使它可以重用。有时，由于操作系统一次同时打开的文件数量有限，因此必须在关闭一个文件后再打开另一个文件。

fclose( )函数的原型如下：

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

其中，fp是由fopen( )返回的文件指针。返回0标志着文件关闭成功。如果关闭失败，则返回EOF。可用标准函数ferror( )（在本章后面讨论）来确定和报告出错消息。通常，fclose( )仅在磁盘从驱动器中过早移走或磁盘上没有更多空间时报错。

9.5.4 写一个字符

C语言的I/O系统定义了两个输出一个字符的等价函数：putc( )和fputc( )。从技术上讲，putc( )通常是作为宏实现的。定义这两种函数是为了保持与旧版本的C兼容，本书只使用putc( )，但如果喜欢，也可使用fputc( )。

函数putc( )把字符写到以前用fopen( )函数以写方式打开的文件中。这个函数的原型是：

```
int putc(int ch, FILE *fp);
```

其中，fp是由fopen( )返回的文件指针，ch是要输出的字符。文件指针告诉putc( )写到哪个磁盘文件中。尽管由于历史原因ch被定义成int，但通常只用其低位字节。

如果putc( )操作成功，则函数返回被输出的字节；否则，返回EOF。

9.5.5 读一个字符

输入一个字符也存在两个等价的函数：getc( )和fgetc( )。定义这两个函数是为了保持与旧版本的C兼容，本书只使用getc( )（它通常是作为宏实现的），但若喜欢，也可使用fgetc( )。

函数getc( )从一个由fopen( )以读方式打开的文件中读取字符，它的原型是：

```
int getc(FILE *fp);
```

其中，fp是由fopen( )返回的FILE类型的文件指针。getc( )返回一个整数，但是字符包含在低位字节中。除非出现错误，否则高位字节为0。

函数getc( )读到文件尾时返回EOF标志。因此要读到一个文本文件的末尾，可以使用下列代码：

```
do {
  ch = getc(fp);
} while(ch!=EOF);
```

如果发生错误，getc( )也返回EOF。可以用ferror( )来确认发生了什么。

9.5.6 使用fopen( ), getc( ), putc( )和fclose( )

函数fopen( )，getc( )，putc( )和fclose( )构成了文件例程的最小集合。下列程序KTOD是使用putc( )，fopen( )和fclose( )的范例。它从键盘上读取字符并把它们写到磁盘文件，直到用户键入一个美元符号“$”为止。文件名在命令行中指定。例如，如果调用程序KTOD，键入KTOD TEST会允许将文本行输入到“TEST”文件中。

```
/* KTOD: A key to disk program. */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  FILE *fp;
  char ch;

  if(argc!=2) {
    printf("You forgot to enter the filename.\n");
    exit(1);
```

```
  }
  if((fp=fopen(argv[1], "w"))==NULL) {
    printf("Cannot open file.\n");
    exit(1);
  }
  do {
    ch = getchar( );
    putc(ch, fp);
  } while (ch != '$');
  fclose(fp);
  return 0;
}
```

辅助程序DTOS读取任何文本文件，并将内容显示在屏幕上。它演示了getc( )。

```
/* DTOS: A program that reads files and displays them
         on the screen. */
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  FILE *fp;
  char ch;
  if(argc!=2) {
    printf("You forgot to enter the filename.\n");
    exit(1);
  }
  if((fp=fopen(argv[1], "r"))==NULL) {
    printf("Cannot open file.\n");
    exit(1);
  }
  ch = getc(fp); /* read one character */
  while (ch!=EOF) {
    putchar(ch); /* print on screen */
    ch = getc(fp);
  }
  fclose(fp);
  return 0;
}
```

现在运行这两个程序，首先使用KTOD创建一个文本文件，然后使用DTOS读其内容。

9.5.7 使用feof( )

如上所述，当遇到文件末尾时，getc( )返回EOF。然而，测试由getc( )返回的值可能不是决定何时到达文件末尾的最好方式。首先，文件系统可以对文本文件和二进制文件进行操作。当一个文件以二进制读方式打开时，可能读到一个与EOF等价的整数值。这样，即使没读到文

件末尾，也可能导致输入例程指示遇到文件结束标志。第二，当 getc( )失败及它到达了文件末尾时，getc( )返回 EOF。仅使用 getc( )的返回值，不可能知道什么发生了。为了解决这个问题，C 文件系统包含了函数 feof( )，这个函数决定何时文件结束。feof( )的原型是：

```
int feof(FILE *fp);
```

若达到文件尾，则函数 feof( )返回真，否则返回 0。因此，下面的例程读一个二进制文件，直到遇到了文件末尾为止：

```
while(!feof(fp)) ch = getc(fp);
```

当然，既可以把这种方法应用于文本文件，也可以应用于二进制文件。

下面的程序用于复制文本文件或二进制文件，它包含一个 feof( )范例。文件以二进制方式打开，feof( )负责检查是否到达文件的末尾。

```
/* Copy a file. */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  FILE *in, *out;
  char ch;

  if(argc!=3) {
    printf("You forgot to enter a filename.\n");
    exit(1);
  }

  if((in=fopen(argv[1], "rb"))==NULL) {
    printf("Cannot open source file.\n");
    exit(1);
  }
  if((out=fopen(argv[2], "wb")) == NULL) {
    printf("Cannot open destination file.\n");
    exit(1);
  }

  /* This code actually copies the file. */
  while(!feof(in)) {
    ch = getc(in);
    if(!feof(in)) putc(ch, out);
  }

  fclose(in);
  fclose(out);

  return 0;
}
```

9.5.8　使用 fputs( )和 fgets( )操作字符串

除 putc( )和 getc( )外，C 文件系统支持另外两个相关函数：fputs( )和 fgets( )。它们从磁盘文件中读或写字符串。这些函数的工作方式与 putc( )和 getc( )相似，但结果不是读（写）一个

字符，而是读（写）一个串。它们的原型如下：

```
int fputs(const char *str, FILE *fp);
char *fgets(char *str, int length, FILE *fp);
```

函数fputs( )把str所指向的字符串写到指定的流中，如果失败，则返回EOF。

函数fgets( )从指定流中读一个字符串，直到读到换行符或读完length-1个字符。如果读到新行，它是原字符串的一部分（不像gets( )那样另起新串），结果字符串将以NULL终止。如果操作成功，则函数返回str，否则返回空指针。

下列程序展示了fputs( )函数的用法。它从键盘中读取字符串，然后写到TEST文件中。键入一个空格键，则程序终止。因为gets( )并不存储换行符，所以在把每个字符串写到文件中之前加入一个换行符以便程序更容易读。

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

int main(void)
{
  char str[80];
  FILE *fp;

  if((fp = fopen("TEST", "w"))==NULL) {
    printf("Cannot open file.\n");
    exit(1);
  }

  do {
    printf("Enter a string (CR to quit):\n");
    gets(str);
    strcat(str, "\n"); /* add a newline */
    fputs(str, fp);
  } while(*str!='\n');

  return 0;
}
```

9.5.9 rewind( )

函数rewind( )把文件位置指针重新置到它的变元所描述的文件起始处。即，它“rewind”该文件。它的原型是：

```
void rewind(FILE *fp);
```

其中，fp是有效的文件指针。

要弄明白rewind( )的例子，可以修改上一节的程序，使文件从创建起便可显示其内容。要完成这一点，在读操作后，程序回到文件起始处，并用fgets( )函数读回文件。注意，文件必须使用“w+”以读/写方式打开。

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

```
int main(void)
{
  char str[80];
  FILE *fp;

  if((fp = fopen("TEST", "w+"))==NULL) {
    printf("Cannot open file.\n");
    exit(1);
  }

  do {
    printf("Enter a string (CR to quit):\n");
    gets(str);
    strcat(str, "\n"); /* add a newline */
    fputs(str, fp);
  } while(*str!='\n');

/* now, read and display the file */
rewind(fp); /* reset file position indicator to
               start of the file. */
  while(!feof(fp)) {
    fgets(str, 79, fp);
    printf(str);
  }

  return 0;
}
```

9.5.10 ferror( )

函数ferror( )确定是否在文件操作期间出错，其原型如下：

int ferror(FILE *fp);

其中，fp是有效的文件指针。在文件操作期间如果出错，则函数返回true，否则返回false。由于每个文件操作均设置错误条件，因而应在每个文件操作后立即调用ferror( )，否则会丢失错误信息。

下面的程序通过从文件中删除制表符并用适当数目的空格代替演示了ferror( )的用法。制表符的尺寸由TAB_SIZE规定。注意，ferror( )函数在每次文件操作后是怎样定义的。为使用该程序，在命令行上指定输入和输出文件的名字。

```
/* The program substitutes spaces for tabs
   in a text file and supplies error checking. */

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

#define TAB_SIZE 8
#define IN 0
#define OUT 1

void err(int e);

int main(int argc, char *argv[])
{
```

```
  FILE *in, *out;
  int tab, i;
  char ch;

  if(argc!=3) {
    printf("usage: detab <in> <out>\n");
    exit(1);
  }

  if((in = fopen(argv[1], "rb"))==NULL) {
    printf("Cannot open %s.\n", argv[1]);
    exit(1);
  }

  if((out = fopen(argv[2], "wb"))==NULL) {
    printf("Cannot open %s.\n", argv[1]);
    exit(1);
  }

  tab = 0;
  do {
    ch = getc(in);
    if(ferror(in)) err(IN);

    /* if tab found, output appropriate number of spaces */
    if(ch=='\t') {
      for(i=tab; i<8; i++) {
        putc(' ', out);
        if(ferror(out)) err(OUT);
      }
      tab = 0;
    }
    else {
      putc(ch, out);
      if(ferror(out)) err(OUT);
      tab++;
      if(tab==TAB_SIZE) tab = 0;
      if(ch=='\n' || ch=='\r') tab = 0;
    }
  } while(!feof(in));
  fclose(in);
  fclose(out);

  return 0;
}

void err(int e)
{
  if(e==IN) printf("Error on input.\n");
  else printf("Error on output.\n");
  exit(1);
}
```

9.5.11 删除文件

函数remove( )删除指定的文件，它的原型是：

```
int remove(const char *filename);
```

如果操作成功，则返回0，否则返回非0值。

下列程序删除命令行中指定的文件。然而，在执行删除操作之前，程序给用户一个改变或确认这种主意的机会，这对计算机用户来说是很有用的。

```
/* Double check before erasing. */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(int argc, char *argv[])
{
  char str[80];

  if(argc!=2) {
    printf("usage: xerase <filename>\n");
    exit(1);
  }

  printf("Erase %s? (Y/N): ", argv[1]);
  gets(str);

  if(toupper(*str)=='Y')
    if(remove(argv[1])) {
      printf("Cannot erase file.\n");
      exit(1);
    }
  return 0;
}
```

9.5.12 清空一个流

函数fflush( )清空一个输出流的内容，其原型为：

```
int fflush(FILE *fp);
```

这个函数将任何缓冲数据的内容写到与fp相关的文件中。如果在fp为空时调用fflush( )，则所有为输出打开的文件被清空。

如果操作成功，fflush( )返回0，否则返回EOF。

9.6 fread( )和fwrite( )

为了读或写一个字节以上的数据类型，C文件系统提供了两个函数：fread( )和fwrite( )，以支持读或写任何数据类型块。fread( )和fwrite( )的原型为：

```
size_t fread(void *buffer, size_t num_bytes, size_t count, FILE *fp);
size_t fwrite(const void *buffer, size_t num_bytes, size_t count, FILE *fp);
```

对 fread( )来说，buffer 是一个指针，它指向一个接收文件中数据的存储区。对 fwrite( )来说，buffer 也是一个指针，它指向要写入文件中的信息块。count 的值指出要读、写多少项（每项长度为 num_bytes）。记住，size_t 类型被定义为某种无符号整数。fp 是一个指向已经打开流的文件指针。

函数 fread( )返回读入的项的数目，如果遇到文件末尾或操作失败，这个值可能少于 count 值。函数 fwrite( )返回写出的项的数目。这个值永远等于 count，除非操作失败。

9.6.1 使用 fread( )和 fwrite( )

只要文件以二进制方式打开，fread( )和 fwrite( )便可以读、写任何类型的信息。例如，下面的程序向磁盘文件上写 double 型，int 型和 long 型变量，然后再读出来。注意如何利用 sizeof 来确定每个数据类型的长度。

```
/* Write some non-character data to a disk file
   and read it back. */
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  FILE *fp;
  double d = 12.23;
  int i = 101;
  long l = 123023L;
  if((fp=fopen("test", "wb+"))==NULL) {
    printf("Cannot open file.\n");
    exit(1);
  }

  fwrite(&d, sizeof(double), 1, fp);
  fwrite(&i, sizeof(int), 1, fp);
  fwrite(&l, sizeof(long), 1, fp);

  rewind(fp);

  fread(&d, sizeof(double), 1, fp);
  fread(&i, sizeof(int), 1, fp);
  fread(&l, sizeof(long), 1, fp);

  printf("%f %d %ld", d, i, l);

  fclose(fp);

  return 0;
}
```

如上述程序所示，缓冲区通常可以是一个保存变量的内存。在本例中，fread( )和 fwrite( )的返回值被忽略了，但在实际应用中，可以检查它们的返回值以寻找错误。

fread( )和 fwrite( )最大的用途之一，就是可以读、写用户自定义的数据类型，特别是结构类型。例如，给出如下结构：

```
struct struct_type {
```

```
  float balance;
  char name[80];
} cust;
```

下面的语句将cust的内容写入由fp指示的文件中。

```
fwrite(&cust, sizeof(struct struct_type), 1, fp);
```

9.7 fseek( )和随机访问I/O

可以用C语言I/O系统中的fseek( )执行随机读写操作，fseek( )可以设置文件位置指针，它的原型如下：

int fseek(FILE *fp, long int *numbytes*, int *origin*);

其中，fp是由fopen( )返回的文件指针，numbytes是从文件的origin位置到当前位置的字节数，origin是下面的宏之一：

origin	宏名
文件开始处	SEEK_SET
当前位置	SEEK_CUR
文件末尾	SEEK_END

因而，若想从文件开始寻找numbytes，则origin取值SEEK_SET；若想从文件当前位置寻找numbytes，则origin取值SEEK_CUR；若想从文件尾寻找numbytes，则origin取值SEEK_END。当文件操作成功时，函数fseek( )返回0，否则，返回非0值。

下面的程序说明了fseek( )，它寻找并显示在指定文件中指定的字节。在命令行上指定文件名和要寻找的字节。

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

int main(int argc, char *argv[])
{
  FILE *fp;

  if(argc!=3) {
    printf("Usage: SEEK filename byte\n");
    exit(1);
  }

  if((fp = fopen(argv[1], "rb"))==NULL) {
    printf("Cannot open file.\n");
    exit(1);
  }

  if(fseek(fp, atol(argv[2]), SEEK_SET)) {
    printf("Seek error.\n");
    exit(1);
  }

  printf("Byte at %ld is %c.\n", atol(argv[2]), getc(fp));
  fclose(fp);
```

```
  return 0;
}
```

可以使用fseek( )来寻找任何数据类型的倍数，方法是用想要的项数乘以数据的长度。例如，假设有个邮件列表具有list_type结构。要寻找包含地址的文件中的第10个地址，使用下面这条语句：

```
fseek(fp, 9*sizeof(struct list_type), SEEK_SET);
```

可以使用ftell( )来决定一个文件的当前位置，它的原型是：

```
long int ftell(FILE *fp);
```

它返回与fp关联的文件的当前位置的地址。如果失败，返回-1。

通常，我们只想要对二进制文件使用随机访问。理由很简单：因为可以对文本文件实行字符转换(translation)，所以在文件中的字节和你想要寻找的字节之间可能没有直接的对应关系。对文本文件，可以使用fseek( )的惟一时间是在寻找以前由ftell( )确定的位置时，此时使用SEEK_SET作为初值。

记住，有一点很重要：如果喜欢，甚至仅包含文本的文件也可以作为二进制文件打开。对包含文本的文件的随机访问，没有固有的限制。限制仅适用于作为文本文件打开的文件。

9.8 fprintf( )和fscanf( )

除已经讨论过的基本I/O函数外，C语言的I/O系统还包括fprintf( )和fscanf( )。这些函数除对文件进行操作外，与printf( )和scanf( )相似。fprintf( )和fscanf( )的原型是：

```
int fprintf(FILE *fp, const char *control_string,. . .);
int fscanf(FILE *fp, const char *control_string,. . .);
```

其中，fp是由fopen( )返回的文件指针。fprintf( )和fscanf( )对fp所指向的文件进行I/O操作。

作为例子，下面的程序从键盘读取一个字符串和一个整数，并把它们写到磁盘文件TEST中，然后再从该文件读出并显示在屏幕上。运行程序以后，检查TEST文件，将会看到，程序包含了可读文本。

```
/* fscanf( ) - fprintf( ) example */
#include <stdio.h>
#include <io.h>
#include <stdlib.h>

int main(void)
{
  FILE *fp;
  char s[80];
  int t;

  if((fp=fopen("test", "w")) == NULL) {
    printf("Cannot open file.\n");
    exit(1);
  }

  printf("Enter a string and a number: ");
```

```
  fscanf(stdin, "%s%d", s, &t); /* read from keyboard */

  fprintf(fp, "%s %d", s, t); /* write to file */
  fclose(fp);

  if((fp=fopen("test","r")) == NULL) {
    printf("Cannot open file.\n");
    exit(1);
  }

  fscanf(fp, "%s%d", s, &t); /* read from file */
  fprintf(stdout, "%s %d", s, t); /* print on screen */

  return 0;
}
```

注意：尽管fprintf( )和fscanf( )是从磁盘文件中读写数据的最容易的方法，但却并不总是最有效的方法。由于格式化的ASCII数据写入文件的格式与在屏幕上显示的相同（而不是二进制方式），因而调用时要引起额外开销。如果要考虑速度与文件长度，最好使用fread( )和fwrite( )。

9.9 标准流

虽然与C文件系统有关，程序一旦开始执行，就有三种流被自动打开。它们是标准输入（stdin）、标准输出（stdout）及标准错误（stderr）。通常，它们对应于控制台，但也可以通过操作系统重定向到支持可重定向I/O环境（如Windows，Unix，OS/2和DOS）的其他设备中。

由于标准流是文件指针，因此它们可使用C语言的I/O系统，以完成控制台上的I/O操作。例如，putchar( )可以定义如下：

```
int putchar(char c)
{
  return putc(c, stdout);
}
```

通常，stdin用于从控制台读，stdout用于向控制台写，stderr也用于向控制台写。可以在使用FILE*类型变量的任何函数中，把stdin，stdout和stderr当作文件指针使用。例如，可以使用fgets( )从控制台输入一个字符串，按如下方式调用：

```
char str[255];
fgets(str, 80, stdin);
```

事实上，按这种方式使用fgets( )是相当有用的。如本书前面提到的，当使用gets( )时，可能越出数组的边界（现在正使用该数组来接收由用户键入的字符），因为gets( )不提供边界检查。当和stdin一起使用时，fgets( )函数提供了一个有用的替代物，因为它能限制所读的字符数，因而防止数组越界。惟一的麻烦是fgets( )没有删除换行符而gets( )删除了，所以你将不得不手工删除它，如下面的程序所示。

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

int main(void)
```

```
{
  char str[80];
  int i;

  printf("Enter a string: ");
  fgets(str, 10, stdin);

  /* remove newline, if present */
  i = strlen(str)-1;
  if(str[i]=='\n') str[i] = '\0';

  printf("This is your string: %s", str);

  return 0;
}
```

记住，stdin，stdout和stderr不是普通变量，也不能通过fopen( )进行赋值。还有，像这些文件指针在程序执行时自动创建一样，它们在程序结束时自动关闭，用户无法随意关闭它们。

9.9.1 控制台I/O连接

实际上，在控制台I/O和文件I/O之间的区别很小。第8章讨论的控制台I/O函数，通过stdin或stdout完成I/O操作。实际上，控制台I/O函数只是一些并行文件函数的特例，它们的存在只是为了方便程序员。

正如上一节描述的那样，可以利用任何文件系统函数来完成控制台I/O。然而，最令人惊奇的还是可以使用控制台I/O函数，如printf( )来完成磁盘文件I/O。这是因为所有的控制台I/O函数都操作于stdin和stdout上。在允许I/O重定向的环境中，这意味着stdin和stdout可对应于包括键盘和屏幕的其他设备。例如，考虑如下程序：

```
#include <stdio.h>

int main(void)
{
  char str[80];

  printf("Enter a string: ");
  gets(str);
  printf(str);

  return 0;
}
```

假定这个程序称为TEST，如果正常执行TEST，则在屏幕上显示提示，从键盘中读一个串，然后将该串显示在屏幕上。然而，在支持I/O重定向的环境中，stdin，stdout或两者都能被重定向到一个文件中。例如，在DOS或Windows环境中，执行TEST如下：

```
TEST > OUTPUT
```

于是TEST的输出结果写到了OUTPUT文件中。若执行TEST如下：

```
TEST < INPUT > OUTPUT
```

stdin被导向INPUT文件中，并发送输出到OUTPUT文件中。

当程序终止时，任何重定向流都复位到它们的默认状态。

9.9.2　使用 freopen( )来重定向标准流

使用函数freopen( )可以重定向标准流。这个函数用于将某一现存流与一个新文件联系起来。因此，可以用它将一个标准流与一个新文件联系起来。它的原型是：

FILE *freopen(const char **filename*, const char **mode*, FILE **stream*);

其中，filename是一个指向你希望与stream指向的流相关联的文件名的指针。该文件以mode值方式打开，与 fopen( )相同。如果操作成功，则返回流，否则，返回 NULL 指针。

下面的程序使用 freopen( )来重定向 stdout 到 OUTPUT 文件。

```
#include <stdio.h>

int main(void)
{
  char str[80];

  freopen("OUTPUT", "w", stdout);

  printf("Enter a string: ");
  gets(str);
  printf(str);

  return 0;
}
```

通常，用freopen( )重定向标准流在一些特殊情况下（例如调试）是很有用的。然而，通过重定向 stdin 和 stdout 来完成磁盘 I/O 不一定与用 fread( )和 fwrite( )函数一样有效。