13 文件

所谓文件，就是指一组相关数据的有序集合，并给这个数据集取一个名称，这个名称即为文件名。文件通常是驻留在外部介质（如硬盘等）上的，在使用时才调入内存中来。

13.0 文件类型

C语言中，从不同的角度可对文件作不同的分类：从用户的角度看，文件可分为普通文件和设备文件；从文件的数据组织方式可以分为文本文件和二进制文件；从对文件的处理方式来看，可以分为缓冲文件系统和非缓冲文件系统两类。

普通文件是指磁盘上或其他外部存储介质上的一个有序数据集。在操作系统中把每一个与主机相连的输入输出设备也当做是一个文件，即设备文件，把它们的输入、输出等同于对磁盘文件的读与写。通常把显示器定义为标准输出文件，键盘被定义为标准输入文件。

文本文件是ASCII码文件，文本文件中存放的字符实际上是每个字符的ASCII码值。文本文件的输入、输出比较方便，但其占用的空间比较多，一个字符占用一个字节。而二进制文件是把内存中的数据的存储形式原样存放在磁盘上，可以节省存储空间。因为不存在转换的操作，对文件的输入、输出操作速度比较快。

缓冲文件系统是指系统在内存中自动为每一个正在使用的文件开辟一个大小固定（一般为512字节）的缓冲区，作为文件与使用数据到内存缓冲区（充满缓冲区），然后再从缓冲区逐个地将数据送给接收变量；向磁盘文件输出数据时，先将数据送到内存中的缓冲区，装满缓冲区后才一起送到磁盘去。用缓冲区可以一次读入一批数据，或输出一批数据，而不是执行一次输入或输出函数就去访问一次磁盘，这样做的目的是减少对磁盘的实际读写次数，因为每一次读写都要移动磁头并寻找磁道扇区，花费一定的时间。非缓冲文件系统不由系统自动设置缓冲区，而由用户自己根据需要设置。在传统的UNIX系统下，用缓冲文件系统来处理文件，用非缓冲文件系统处理二进制文件。1983年ANSI C标准决定不采用非缓冲文件系统，而只采用缓冲文件系统。即用缓冲文件系统处理文本文件，也用它来处理二进制文件。也就是将缓冲文件系统扩充为可以处理二进制文件。

一般把缓冲文件系统的输入输出称为标准输入输出（标准I/O），非缓冲文件系统的输入输出称为系统输入输出（系统I/O）。在C语言中，没有输入输出语句，对文件的读写都是用库函数来实现的。ANSI C规定了标准输入输出函数，用它们对文件进行读写。这节主要探讨ANSI C的文件系统以及对其的读写方法。

13.1 与文件相关的指针

缓冲文件系统中，每个被使用的文件都在内存中开辟一个区，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）。这些信息保存在一个结构体类型的变量中，该结构体类型是由系统定义的，取名为FILE，其形式为：

```
typedef struct
{
    ...  // 结构体成员项，用来存放文件信息
} FILE;
```

对于用户而言，在编写源程序时不必关心FILE结构的细节，只要知道对于每一个要操作的文件，都必须定义一个指针变量，即文件指针。通过文件指针，就可以对它所指的文件进行各种操作。文件指针的定义形式为：

```
FILE *<指针变量名>;
```

例如：`FILE *fp;`

定义文件指针fp为指向FILE类型结构体的变量指针。程序中对文件的操作，必须通过指向文件的文件指针变量来完成。一般来说，有n个文件就要定义n个FILE类型的指针变量。

13.2 文件的打开与关闭

在对文件进行读写之前必须先打开文件，即建立文件的各种有关信息，使文件指针指向该文件。文件使用结束后，应及时关闭文件。

13.2.0 文件的打开

文件的打开，可以用`fopen()`函数来实现，函数调用的一般形式为：

`文件指针名 = fopen(文件名,使用文件方式);`

其中，文件指针名必须为被定义为FILE类型的指针变量，文件名是被打开文件的文件名，使用文件方式是指文件的类型和操作要求。文件名是字符串常量或字符串数组。

```
// 例如:
FILE *fp;
fp = fopen("D:\\filename.txt","w+");
```

表示打开D盘下的文件filename.txt（如果没有则创建），使文件指针fp指向它，并允许对其进行读写操作。注意：在打开文件时，如果要指定文件的路径需要用双反斜杠"\\"符号。如果不指定文件路径，如直接写"filename.txt"，则fopen()函数处理文件的路径默认为当前工作目录。

常见的使用文件方式见下表。

使用文件方式	含义
"r"	打开一个文本文件，只允许读数据
"w"	打开或建立一个文本文件，只允许写数据
"a"	打开一个文本文件，并在文件末尾写数据
"rb"	打开一个二进制文件，只允许读数据
"wb"	打开或建立一个二进制文件，只允许写数据
"ab"	打开一个二进制文件，并在文件末尾写数据
"r+"	打开一个文本文件，允许读写数据
"w+"	打开或建立一个文本文件，允许读写数据
"a+"	打开一个文本文件，允许读或在文件末写数据
"rb+"	打开一个二进制文件，允许读写数据
"wb+"	打开或建立一个二进制文件，允许读写数据
"ab+"	打开一个二进制文件，允许读或在文件末写数据

表 13 - 1 常用的使用文件方式

当文件被正常打开时，函数返回值为指向文件结构体的指针，否则返回值为NULL。通常在执行程序文件的其他操作之前，要先确认fopen()函数执行是否成功，因此，常用以下程序段打开文件：

```
FILE *fp;
if ((fp = fopen("filename.dat","rb")) == NULL) {
    printf("Can not open file!\n");
    exit(1);
}
```

这段程序的意思是，如果返回的指针为空，则表示不能打开指定的文件，并给出提示信息，exxit()函数的作用是关闭所有文件，然后退出程序。

13.2.1 文件的关闭

文件一旦使用完毕，应及时关闭文件，以免发生文件数据的丢失等问题。因为，在向文件写数据时，是先将数据输到缓冲区，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，就会导致缓冲区中的数据丢失。用fclose()函数关闭文件，可以避免这个问题，它先把缓冲区中的数据输出到磁盘文件后才释放文件指针变量。

fclose()函数的调用形式为：

`fclose(文件指针);`

例如：`fclose(fp);`

fclose()函数也返回一个值：当成功执行了关闭操作，返回值为0；如果返回值为非零值，则表示文件关闭出错。

13.3 文件的读写

文件被成功打开后，便可以对它进行读写操作了。常用的文件读写操作有如下几种：

1. 针对一个字符的读写；
2. 针对一个字符串的读写；
3. 针对一个数据块的读写；
4. 带格式的读写。

13.3.0 读写字符函数

读写字符函数是以字符（字节）为单位的读写函数。每次可向文件输出或从文件读入一个字符。字符输入和输出函数分别为fgetc()和fputc()函数，它们的原型分别为：

```
int fputc(int ch, FILR *fp);
int fgetc(FILE *fp);
```

1. fputc()函数

该函数的功能是把一个字符输出到指定文件中，函数调用的形式为：

`fputc(ch,fp);`

其中，整型量ch为要输出的字符，它可以是整型或字符型，但不管是什么类型，在执行fputc()函数输出到文件时都转换成字符型进行输出；fp是指定文件的文件指针变量。所指向的文件必须是可写方式打开的。

fputc()函数有一个返回值，如果输出成功则将所输出的字符转换成整型数据返回，否则返回一个EOF值。EOF是在头文件stdio.h中定义的符号常量，值为-1 。

2. fgetc()函数

该函数的功能是从指定的文件中读入一个字符，函数调用的形式为：

`ch = fgetc(fp);`

其中，fp是指定文件的文件指针变量，所指向的文件必须是可读方式打开的。ch为整型变量，用于保存fgetc(fp)返回的一个字符。

调用一次fgetc()函数后，文件内部的位置指针将向后移动一个字节的位置，因此可以通过连续多次调用fgetc()函数来读取多个字符。应注意文件指针和文件内部的位置指针不是一回事。文件指针是指向整个文件的，须在程序中定义说明，只要不重新赋值，文件指针的值是不变的。文件内部的位置指针用以指示文件内部的当前读写位置，每读一次，该指针均先后移动，它不需在程序中定义说明，而是由系统自动设置的。如果在读取字符时遇到文件结束符，fgetc()函数返回一个文件结束标志EOF，可以用它来判断是否读到了文件的末尾。

在C语言中，提供了一个专门的feof()函数来判断文件是否真的结束。例如，一般将顺序读取文件数据的程序段写为：

```
while (!feof(fp)) {
    ch = fgetc(fp);
    ... ...
}
```

当未遇到文件结束符时，feof(fp)的值为0，则!feof(fp)为1，读取一个字节的数据赋值，之后循环再判断feof(fp)的值，直到遇到文件结束符，此时，feof(fp)的值为1，则!feof(fp)为0，while循环结束。

```
#include <stdio.h>
#include <stdlib.h>
// 从键盘输入一行字符，将字符按顺序输出到test.txt文件中，然后读取文件内容，将其输出至显示器
int main()
{
    FILE *fp;
    int ch;

    if ((fp = fopen("test.txt","w+")) == NULL) {
        printf("Cannot open the file!");
        exit(1);
    }
    printf("Input a string:\n");
    ch = getchar();
    while (ch! = '\n') {
        fputc(ch,fp);
        ch = getchar();
    }
    fclose(fp);
    if ((fp = fopen('test.txt','r')) == NULL) {
        print("Cannot open the file!");
        exit(1);
    }
    printf("Output the string:\n");
    ch = fgetc(fp);
    while (ch != EOF) {
        putchar(ch);
        ch = fgetc(fp);
    }
```

```
    print("\n");
    fclose(fp);
    /* 运行结果:
    Input a string:
    Hello world!
    Output the string:
    Hello world!
    */
    return 0;
}
```

程序中出现两次打开和关闭打开和关闭文件的语句，这是因为在顺序存取文件的方法下，随着将字符输出到文件，输出操作结束时，文件内部的位置指针同时指向了文件末尾，如果不将位置指针到文件开头，则不能正确读取相应文件的内容。因此，在此例中是先关闭文件然后再重新打开文件，让位置指针自动指向文件开始位置。

另外，表 13 - 2 中两段代码的作用是一样的，都是判断文件中位置指针是否已经指向文件末尾，一般可以相互替换。但是，C99标准认为feof()函数不安全，建议大家使用第一种方式判断是否到了文件末尾。

第一种	第二种

`ch = fgetc(fp);`
`while (ch != EOF) {`
 	
putchar(ch);
ch = fgetc(fp);
}
	`while (!feof(fp)) {`
	ch = fgetc(fp);
	putchar(ch);
	}

表 13 - 2 判断文件末尾的两种方式

13.3.1 读写字符串函数

读写字符串函数是实现整行文本读写的函数。在字符串的输入输出内容中介绍了函数gets()和puts()，它们分别表示从标准输入设备输入字符串和输出字符串至标准设备为操作对象，而可以是任意的普通文件。fgets()和fputs()的函数原型分别为：

```
    char *fgets(char *buffer, int n, FILE *fp);
    int fputs(const char *buffer,FILE *fp);
```

1. fgets()函数

fgets()函数是从指定的文件中读取一个字符串到字符数组中，其调用形式为：

`fgets(字符数组名, n, 文件指针);`

其中，n是一个正整数，表示从指定文件中读取的字符串不超过n-1个字符；在读入的最后一个字符后加上字符串标志'\0'，放入给定的字符数组里。

例如：`fgets(str, n, fp);`

表示从fp所指的文件中读取n-1个字符存放到数组str中。

fgets()函数的使用需要注意以下几点：

- 如果在未读满n-1个字符时，已读取到一个换行符或一个EOF（文件结束标志），则结束本次操作。
- 读取成功后，fgets()函数返回字符数组的首地址，否则返回空指针。

2. fputs()函数

fputs()函数的功能是向指向的文件输出一个字符串，其调用形式为：

`fputs(字符串, 文件指针);`

其中，字符串可以是字符串常量，也可以是字符数组名或指针变量。字符串末尾的'\0'不输出。若输出成功，函数返回值为0，否则为EOF。

例如：`fputs("123456", fp);`

表示把字符串"123456"输出到文件指针fp所指的文件中。

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // 新建一个文件，输出一个字符串到该文件中，然后再读取文件中的部分字符串，并将其输出至显示器
    FILE *fp;
    char ch[80];
    if ((fp = fopen("test.txt", "w")) == NULL) {
        printf("Cannot open the file!");
        exit(1);
    }
    printf("Input a string:\n");
    scanf("%s", ch);
    fclose(fp);
    // 從文件中讀出字符串顯示在顯示器上
    if ((fp = fopen("test.txt", "r")) == NULL) {
        printf("Cannot open the file!");
        exit(1);
    }
    printf("Output a string:\n");
    fgets(ch, 7, fp);
    printf("%s", ch);
    printf("\n");
    fclose(fp);
    /* 运行結果:
    Input a string:
    I love china!
    Output a string:
    I
    */
    return 0;
}
```

13.3.2 格式化读写函数

格式化读写函数：fscanf() 和 fprintf() 函数，与前面使用的 scanf() 函数和 printf() 函数的功能相似，都是进行格式化读写。两者的区别在于 fscanf() 和 fprintf() 函数的读写对象不是键盘和显示器，而是磁盘文件。这两个函数名的第一个字符是"f"，表示是对文件 ("file")的操作。这两个格式化读写函数的函数原型分别为：

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

它们的一般调用形式分别为：

```
    fscanf(文件指针, 格式控制字符串, 输入项表);
    fprintf(文件指针, 格式控制字符串, 输出项表);
    // 例如:
    fscanf(fp, "%d %d", &i, &j);
```

表示从 fp 所指向的文本文件总读入两个整型数放入变量 i 和 j 中。注意，文件中的两个整数之间一般要用空格（或跳格符、回车符）隔开。因为，fscanf() 函数再读取数据时，它将空格（或跳格符、回车符）当做数据间的分隔符号。如果函数读入出错，则返回 EOF 值；否则返回格式正确读入数据项的个数。

`fprintf(fp, "%d %c", x, y);`

表示将变量 x 中的整数和变量 y 中字符分别按照 %d 和 %c 格式输出到 fp 所指向的文本文件中

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // 从键盘输入字符型、整型、实型数各一个，按格式输出到文件中，再按格式从文件中读取相应数据
    FILE *fp;
    int i1, i2;
    char ch1, ch2;
    float a1, a2;

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

    printf("从键盘输入字符型、整型、实型数各一个，按格式存入文件: \n");
    scanf("%c %d %f", &ch1, &i1, &a1);
    fprintf(fp, "%c %5d %5.2f", ch1, i1, a1);
    fclose(fp);

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

    printf("从文件中按格式读取字符型、整型、实型数: \n");
    fscanf(fp, "%c %d %f", &ch2, &i2, &a2);
    printf("%c %5d %5.2f\n", ch2, i2, a2);
    fclose(fp);
    /* 运行结果:
    从键盘输入字符型、整型、实型数各一个，按格式存入文件:
    f 4 5.42
    从文件中按格式读取字符型、整型、实型数:
    f     4  5.42
    */
    return 0;
}

```

13.3.3 读写数据块函数

C语言还提供了用于整块数据的读写函数：fread() 和 fwrite() 函数。这样，不局限于每次读写一个字符或者一行字符串，可用来读写一组数据，如一个数组元素，一个结构变量的值等。它们可以用来读、写二进制文件和文本文件，但主要是用于读写二进制文件。如果文件以二进制形式打开，用 fread() 和 fwrite() 函数可以读写任何类型的信息。

1. fread() 函数

其调用形式为：

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

其中，buffer 是一个指针，它表示从文件指针 fp 指向的文件中读取一批数据后所存放空间的首地址；size表示所读取一个数据块的字节数，count 表示要读取的数据块的各数。

例如：`fread(a, 4, 5, fp);`

其中，a 是一个整型数组名，一个整型变量占4个字节（也可用 sizeof(int) 表示）。fread() 函数返回实际读入的元素个数，在读取的同时还应当注意是否读取到了文件末尾。

2. fwrite() 函数

其调用形式为：

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

其中，buffer 是一个指针，它指向存放着一批数据的空间的首地址。即将 buffer 所指向的一批数据输出到 fp 所指向的文件中。size 表示写入一个数据块的字节数，count 表示要写入的数据块的个数。

例如：`fwrite(a, 4, 5, fp);`

其中，a 是一个数组名，一个整型变量占4个字节（也可以用 sizeof(int) 表示）。fwrite() 函数将数组 a 中的5个元素的数据（5 * sizeof(int) 个字节）输出到 fp 所指向的文件中。

fwrite() 函数返回实际输出的数据元素个数，如果这个数小于 count ，就说明函数执行中出现了错误。

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // 从键盘输入字符型、整型、实型数各一个，按格式输出到文件中，再按格式从文件中读取相应数据
    FILE *fp;
    int i;
    float b[8], a[8]={12.3, 5.8, 16.8, -60.5, 0.98, 3.142, -2.71, 108};

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

    printf("将数组 a 中的如下数据写入文件中: \n");
    for (i=0; i<8; ++i) { printf("%8.3f", a[i]);}
    fwrite(a, sizeof(float), 8, fp);
    fclose(fp);

    if ((fp = fopen("test.txt", "rb")) == NULL) {
        printf("Cannot open the file!");
        exit(1);
```

```
    }

    printf("\n读取文件中的前4个数据:\n");
    fread(b, sizeof(float), 4, fp);
    for (i=0; i<4; ++i) { printf("%8.3f", b[i]);}
    printf("\n");
    fclose(fp);
    /* 运行结果:
    将数组 a 中的如下数据写入文件中:
      12.3000   5.8000  16.800 -60.500   0.980   3.142  -2.710 108.000
    读取文件中的前4个数据:
      12.3000   5.8000  16.800 -60.500
    */
    return 0;
}
```

13.4 文件指针的定位

当通过 fopen() 函数打开文件时，文件位置指针总是指向文件的开头第一个数据之前。而前面介绍的对文件的读写方式都是顺序读写，因此，读写文件只能从头开始，顺序读写各个数据。但在实际问题中常要求只读写文件中某一指定的部分。为了解决这个问题可移动文件内部的位置指针到需要读写的位置，在进行读写，这种读写称为随机读写。实现随机读写的关键是要按要求移动位置指针，这称为文件的定位。文件定位移动文件内部位置指针的函数只要有3个，即 fseek() 函数、ftell() 函数和 rewind() 函数。

13.4.0 文件位置指针定位函数

fseek()函数的作用是移动文件位置指针到指定的位置上，这样，可以实现从指定的位置开始读或写操作的功能。该函数的原型为：

```
int fseek(FILE *fp, long offset, int origin);
```

其中，fp 为文件指针；offset 是以字节为单位的位移量，为长整型数，以便在文件长度大于64kB时不会出错，该取值可以取负数，对于二进制文件，当位移量为正整数时表示从当前位置向文件末尾方向移动相应的距离，当为负时表示从当前位置向文件头方向移动相应的距离，当用常量表示移动量时需要加后缀“L”。origin 表示起点位置，指定位移量是以它做参考点的，起始点既可以用标识符来表示，也可用数字来代表。规定的起始点有3种：文件首、当前位置和文件尾。其表示方法如下表所示。

起始点	宏	宏值
文件首	SEEK_SET	0
文件当前位置	SEEK_CUR	1
文件尾	SEEK_END	2

表 13 - 3 文件起始点

```
    // 例如:
    fseek(fp, 10L, SEEK_SET);  // 将位置指针移动到离文件首10个字节处
    fseek(fp, 10L, SEEK_CUR);  // 将位置指针移动到离当前位置10个字节处（向后移动）
    fseek(fp, -10L, SEEK_END); // 将位置指针移动到离文件尾10个字节处
```

fseek() 函数一般用于二进制文件。对于文本文件，因为要发生字符转换，计算位置时往往发生混乱。函数定位成功时，返回值为0，否则返回一个非零的值。

13.4.1 文件位置指针获取函数

ftell() 函数的作用是获得位置指针所指的当前位置，用相对于文件首的位移量来表示。由于文件中的位置指针经常移动，往往不容易知道其当前位置，利用 ftell() 函数可以得到当前位置。该函数的原型为：

`long int ftell(FILE *fp);`

如果 ftell() 函数执行成功，返回位置指针所指的当前位置，否则返回 -1L。例如：

```
    i = ftell(fp);
    if (i == -1L) {printf("ERROR!\n");}
```

长整型变量 i 用于存放当前位置（相当于文件首的位移量），如果出错则 i 的值为 -1L。

13.4.2 文件位置指针归位函数

rewind() 函数的作用是使位置指针重新指向文件首。该函数的原型为：

`void rewind(FILE *fp);`

rewind() 函数没有返回值。其作用等价于下面的语句。

`(void) fseek(fp, 0L, SEEK_SET);`

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // 读取文件中的后4个数据，再读取前4个数据
    FILE *fp;
    int i;
    float b[8], a[8] = {12.3, 5.8, 1.68, -60.5, 0.98, 3.142, -2.71, 108};

    if ((fp = fopen("text.txt", "wb")) == NULL) {printf("Cannot open the file!"); exit(1);}
    printf("将数组 a 中的如下数据写入文件中: \n");
    for (i=0; i<8; ++i) {
        printf("%8.3f", a[i]);
    }
    fwrite(a, sizeof(float), 8, fp);
    fclose(fp);

    if ((fp = fopen("text.txt", "rb")) == NULL) {printf("Cannot open the file!"); exit(1);}
    printf("\n先读取文件中的后4个数据: \n");
    fseek(fp, -sizeof(float)*4, SEEK_END);
    fread(b, sizeof(float), 4, fp);
    for (i=0; i<4; ++i) {
        printf("%8.3f", b[i]);
    }
    printf("\n再读取文件中的前4个数据: \n");
    rewind(fp);
    fread(b, sizeof(float), 4, fp);
    for (i=0; i<4; ++i) {
        printf("%8.3f", b[i]);
    }
    printf("\n");
    fclose(fp);
```

```
    /* 运行结果:
    将数组 a 中的如下数据写入文件中:
      12.300   5.800  16.800 -60.500   0.980   3.142  -2.710 108.000
    先读取文件中的后4个数据:
       0.980   3.142  -2.710 108.000
    再读取文件中的前4个数据:
      12.300   5.800  16.800 -60.500
    */
    return 0;
}

```