C 语言

第 14 讲、文件操作

张晓平

武汉大学数学与统计学院

2017 年 5 月 31 日

1. 文件类型

2. 文件的打开与关闭

3. 文件的读写

4. 文件的定位

5. 出错检测

1. 文件类型

文件类型

所谓“文件”是指一组相关数据的有序集合。

- 数据以文件的形式存放在外部介质 (一般是磁盘、磁带、光盘等) 上，在操作系统中是以文件为单位对数据进行管理的。
- 以文件名作为访问文件的标识。

文件类型

C 语言把文件看作一个字节序列，即由一连串的字节组成。根据文件中的数据组织形式，数据文件可分为 ASCII 码文件和二进制文件。

- ASCII 码文件，也称“文本文件”，其每一个字节存放一个 ASCII 码。
- 二进制文件，把内存中的数据按其在内存中的存储形式存放在磁盘上。

文件类型

例：十进制整数 10000，在内存中占两个字节，其存放形式是：00100111 00010000。

- 在二进制文件中也按这种方式存放。
- 在 ASCII 文件中，十进制整数 10000 存放为 31H、30H、30H、30H、30H，占五个字节，它们分别是 1、0、0、0、0、0 字母的 ASCII 码。

文件类型

按照操作系统对磁盘文件的读写方式，文件可以分为“缓冲文件系统”和“非缓冲文件系统”。

文件类型

定义 (缓冲文件系统) 操作系统在内存中为每一个正在使用的文件开辟一个读写缓冲区。

- 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上。
- 如果从磁盘向内存读入数据，则一次从磁盘文件将一批数据输入到内存缓冲区，然后再从缓冲区逐个地将数据送到程序数据区。

定义 (非缓冲文件系统) 操作系统不自动开辟确定大小的读写缓冲区，而由程序为每个文件设定缓冲区。

文件类型

- 在 UNIX 系统下，用缓冲文件系统来处理文本文件，用非缓冲文件系统处理二进制文件。
- ANSI C 标准只采用缓冲文件系统。

文件类型

缓冲文件系统中，每一个使用的文件都在内存中开辟一个“文件信息区”，用来存放文件的相关信息（文件的名字、文件当前的读写位置、文件操作方式等）。这些信息保存在一个结构体变量中，该结构体是由系统定义的，取名为 `FILE`。

文件类型

在 stdio.h 中，有以下文件类型的声明：

```
typedef struct {
  int level;                  // 缓冲区“满”或“空”的程度
  unsigned flags;             // 文件状态标志
  char fd;                    // 文件描述符
  unsigned char hold;         // 如无缓冲区不读取字符
  int bsize;                  // 缓冲区的大小
  unsigned char * buffer;     // 数据缓冲区的位置
  unsigned char * curp;       // 指针，当前的指向
  unsigned istemp;            // 临时文件，指示器
  short token;                // 用于有效性检查
} FILE;
```

文件类型

定义文件指针变量的一般形式为：

```
FILE *文件结构指针变量名
```

例如：

```
FILE * fp;
```

注意：只有通过文件指针，才能调用相应的文件。

2. 文件的打开与关闭

文件的打开与关闭

文件操作的过程：对磁盘文件的操作必须“先打开，后读写，最后关闭”。

“打开”文件的含义：以某种方式从磁盘上查找指定的文件或创建一个新文件。

ANSI C 规定了标准输入输出函数库，用 `fopen()` 打开文件。

文件的打开与关闭

`fopen()` 的函数原型声明为

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

参数说明

- 参数 `path` 字符串包含欲打开的文件路径及文件名;
- 参数 `mode` 字符串则表示“文件打开方式”。

文件的打开与关闭

例如，

```
FILE * fp;
fp = fopen("file1", "r");
```

表示要打开名字为 `file1` 的文件，使用文件方式为“只读”。

- 如果打开成功，返回一个指向 `file1` 文件的指针；
- 如果打开失败，返回一个 `NULL` 指针。

文件的打开与关闭

表: 使用文件方式

"r"（只读）	为输入打开一个文本文件
"w"（只写）	为输出打开一个文本文件
"a"（追加）	为追加打开一个文本文件
"rb"（只读）	为输入打开一个二进制文件
"wb"（只写）	为输出打开一个二进制文件
"ab"（追加）	为追加打开一个二进制文件

文件的打开与关闭：文件的打开 (fopen)

表: 使用文件方式

"r+"（读写）	为读/写打开一个文本文件
"w+"（读写）	为读/写创建一个文本文件
"a+"（读写）	为读/写打开一个文本文件
"rb+"（读写）	为读/写打开一个二进制文件
"wb+"（读写）	为读/写创建一个二进制文件
"ab+"（读写）	为读/写打开一个二进制文件

文件的打开与关闭

- 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。不能用"r" 方式打开一个不存在的文件（即输入文件），否则出错。
- 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，则在打开时新建一个文件；如果该文件存在，则先删除该文件，然后重新建立一个新文件。

文件的打开与关闭

- 如果希望向文件尾添加新数据（不删除原有数据），则应该用"a" 方式打开。但要求此时文件必须存在，否则出错。
- 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。

文件的打开与关闭

- 如果不能实现打开文件的任务，fopen() 将带回一个出错信息。用带"r" 的方式（"r"、"rb"、"r+"、"rb+"）打开文件时，若文件不存在，则返回 NULL 指针。常用以下方式打开文件：

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

文件的打开与关闭

在读取文本文件时，将回车换行符转换为一个换行符；在写入文本文件时把换行符转换为回车和换行两个字符。在用二进制文件时，不进行转换。

文件的打开与关闭：文件的关闭 (fclose)

在使用完一个文件后应该关闭它，“关闭”文件就是使文件指针与文件脱离，此后不能再通过该指针对原来与其相联系的文件进行读写操作。应养成在程序终止前关闭所有文件的习惯。

文件的打开与关闭

可以用 fclose() 关闭文件，其函数原型声明为

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

fclose() 也带回一个返回值。当顺利关闭文件时，返回 0；否则返回 EOF(-1)。

文件的打开与关闭

例　编制程序，统计某文本文件中的字符数。

文件的打开与关闭 I

```
1  /* count.c -- using standard I/O */
2  #include <stdio.h>
3  #include <stdlib.h> // ANSI C exit() prototype
4
5  int main(int argc, char *argv[])
6  {
7    int ch;
8    // place to store each character as read
9    FILE *fp;
10   // "file pointer"
11   long count = 0;
12
13   if (argc != 2) {
```

文件的打开与关闭 II

```
14      printf("Usage: %s filename\n", argv[0]);
15      exit(1);
16    }
17
18    if ((fp = fopen(argv[1], "r")) == NULL) {
19      printf("Can't open %s\n", argv[1]);
20      exit(1);
21    }
22
23    while ((ch = getc(fp)) != EOF) {
24      putc(ch,stdout); // same as putchar(ch);
25      count++;
26    }
27    fclose(fp);
```

文件的打开与关闭 III

```
28    printf("File %s has %ld characters\n", argv[1],
      count);
29
30    return 0;
31  }
```

3. 文件的读写

文件的读写

当文件打开后，就可以对它进行读写了。常用的读写函数有：

- `fputc()` 和 `fgetc()`
- `fprintf()` 和 `fscanf()`
- `fread()` 和 `fwrite()`

文件的读写：fputc()

对于 fputc()，

- 原型声明

```
int fputc(int c, FILE * stream);
```

- 函数说明：fputc() 将参数 c 转为 unsigned char 后写入参数 stream 指定的文件中。
- 返回值：成功时返回字符 c 的 ASCII 码，失败时返回 EOF。

文件的读写：fgetc()

对于 fgetc()，

- 原型声明

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

- 函数说明：fgetc() 从参数 stream 所指的文件中读取一个字符。若读到文件尾而无数据时便返回 EOF.

- 对于文本文件，遇文件尾时返回文件结束标志 EOF。
- 对于二进制文件，用 feof(fp) 判别是否遇文件尾，feof(fp)==1 说明遇文件尾。

文件的读写：fgetc()

例　从文本文件中顺序读入文件内容，并在屏幕上显示出来。

文件的读写：fgetc() I

```
1  // file2screen.c:
2  #include<stdio.h>
3  #include<stdlib.h>
4
5  int main(int argc, char * argv[])
6  {
7    FILE * fp;
8    int ch;
9
10   if (argc != 2) {
11     printf("Usage: %s filename\n", argv[0]);
12     exit(1);
13   }
```

文件的读写：fgetc() II

```
14
15    if ((fp = fopen(argv[1], "r")) == NULL) {
16      printf("Can't open %s\n", argv[1]);
17      exit(1);
18    }
19
20    while ((ch = fgetc(fp)) != EOF) {
21      putchar(ch); // same as putchar(ch);
22    }
23
24    return 0;
25  }
```

文件的读写：fgetc()

从二进制文件中顺序读入文件内容，可以用：

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

这种方法也适用于文本文件。

文件的读写：fputc() 和 fgetc() 的应用举例

例　从键盘输入一些字符，逐个把它们送入磁盘文件，直到从键盘输入 # 为止。

文件的读写：fputc() 和 fgetc() 的应用举例 I

```
1  // screen2file.c:
2  #include <stdio.h>
3  #include <stdlib.h>
4  int main(void)
5  {
6    char ch;
7    FILE *fp;
8
9    if ( (fp = fopen("file1.txt", "w")) == NULL) {
10     printf("Cannot open file1.txt!\n");
11     exit(1);
12   }
13   while ((ch = getchar()) != '#')
```

文件的读写：fputc() 和 fgetc() 的应用举例 II

```
14      fputc(ch, fp);
15    fclose(fp);
16
17    return 0;
18  }
```

文件的读写：fputc() 和 fgetc() 的应用举例

例　将一个磁盘文件的内容复制到另一个磁盘文件。

文件的读写：fputc() 和 fgetc() 的应用举例 I

```
1  // copy.c:
2  #include <stdio.h>
3  #include <stdlib.h>
4  int main(int argc, char * argv[])
5  {
6    char ch;
7    FILE * fsrc, * fdes;
8
9    if (argc < 3) {
10     printf("Usage: %s srcfile desfile\n", argv[0]);
11     exit(1);
12   }
13
```

文件的读写：fputc() 和 fgetc() 的应用举例 II

```
14
15    if ((fsrc = fopen(argv[1], "r")) == NULL ||
16        (fdes = fopen(argv[2], "w")) == NULL) {
17      printf("Cannot open files!\n");
18      exit(1);
19    }
20
21    while ((ch = getc(fsrc)) != EOF)
22      fputc(ch, fdes);
23    printf("Successed copy srcfile to desfile\n");
24
25    fclose(fsrc);
26    fclose(fdes);
27
```

文件的读写：fputc() 和 fgetc() 的应用举例 III

```
28    return 0;
29 }
```

文件的读写：fscanf() 和 fprintf()

fprintf()、fscanf() 与 printf()、scanf() 的作用相仿，都是格式化读写函数。fprintf() 和 fscanf() 的读写对象是磁盘文件，而 printf() 和 scanf() 的读写对象是终端。

文件的读写：fscanf() 和 fprintf()

它们的函数原型为：

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

除增加“文件指针”外，其他与 printf()/scanf() 用法相同。

文件的读写：fscanf() 和 fprintf()

例如：

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

它的作用是将整型变量 i 和浮点型变量 t 的值按%d 和%6.2f 的格式输出到 fp 所指向的文件中。

如果 i=3, t=4.5，则输出到磁盘文件上的是以下字符串：

```
3, 4.50
```

文件的读写：fscanf() 和 fprintf()

同样，用 fscanf() 可以从磁盘文件上读入 ASCII 字符：

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

磁盘文件上如果有以下字符：

```
3, 4.5
```

则将磁盘文件的数据 3 送给变量 i，4.5 送给变量 t。

文件的读写：fscanf() 和 fprintf() 的应用举例

例　编制程序，向磁盘文件添加单词，并显示文件内容在屏幕上。

文件的读写：fscanf() 和 fprintf() 的应用举例 I

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX 40
4  int main(void)
5  {
6    FILE *fp;
7    char words[MAX];
8
9    if((fp = fopen("words", "a+")) == NULL) {
10     fprintf(stdout, "Can't open \"words\" file.\n");
11     exit(1);
12   }
13
```

文件的读写：fscanf() 和 fprintf() 的应用举例 II

```
14    puts("Enter words to add to the file: press the
      Enter ");
15    puts("key at the beginning of a line to terminate."
      );
16    while ( gets(words) != NULL && words[0] != '\0')
17      fprintf(fp, "%s", words);
18    puts("File contents:");
19    rewind(fp);
20    while(fscanf(fp, "%s", words)==1)
21      puts(words);
22    fclose(fp);
23
24    return 0;
25  }
```

文件的读写：fscanf() 和 fprintf() 的应用举例 III

文件的读写：fscanf() 和 fprintf()

用 fprintf() 和 fscanf() 对磁盘文件操作，由于在输入时要将 ASCII 码转换为二进制形式，在输出时又要将二进制转换为字符，花费时间比较多，因此，在内存与磁盘频繁交换数据的情况下，最好不用 fprintf() 和 fscanf()，而用 fread() 和 fwrite()。

文件的读写：putw() 和 getw()

putw() 和 getw() 用来对磁盘文件读写一个字 (整数)。例如：

```
putw(10, fp);      /* 整数 10 写入文件 fp */
i = getw(fp);      /* 从文件 fp 读一个整数给变量 i */
```

文件的读写：fgets() 和 fputs()

fgets() 的作用是从指定文件读入一个字符串。例如：

```
fgets(str, n, fp); // 从文件 fp 读 n-1 个字节到 str
                   // str 最后一个字节加'\0'
```

fputs() 的作用是向指定的文件输出一个字符串。例如：

```
fputs(str, fp);    // 把字符串 str 写入 fp
```

文件的读写：fread() 和 fwrite()

以下代码将 num 作为一个含 8 个字符的字符串 0.333333 存储

```
double num = 1./3.;
fprintf(fp,"%f", num);
```

- 使用%.2 可以把它存储为含 4 个字符的字符串 0.33;
- 使用%.12f 可把它存储为含 14 个字符的字符串 0.333333333333。

改变占位符可以改变存储这一值所需的空间大小，也会导致存储不同的值。

文件的读写：fread() 和 fwrite()

在 num 的值存为 0.33 以后，读取文件时就没法恢复其精度。总之，fprintf() 以一种可能改变数字值的方式将其转换为字符串。

文件的读写：fread() 和 fwrite()

- 最精确的存储数字的方法就是使用与程序所使用的相同的位格式。故一个 double 值应该存储在一个 double 大小的单元中。
- 如果把数据存储在一个使用与程序具有相同表示方法的文件中，就称数据以二进制形式存储。这中间没有从数字形式到字符串形式的转换。
- fread() 和 fwrite() 提供了这种二进制服务，用来读写一个数据块。

文件的读写：fread() 和 fwrite()

fwrite() 将二进制数据写入文件，其原型为

```
size_t fwrite(const void * ptr, size_t size, size_t
nmemb, FILE * fp);
```

其中：

- 指针 ptr 是要写入的数据块的地址。
- size 表示要写入的数据块的大小（以字节为单位）。
- nmemb 表示数据块的数目。
- fp 指定要写入的文件
- 返回成功写入的项目数。正常情况下，它与 nmemb 相等，若有错，返回值会小于 nmemb。

文件的读写：fread() 和 fwrite()

要保存一个 256 字节大小的数据对象（如一个数组），可以这么做

```
char buffer[256];
fwrite(buffer, 256, 1, fp)
```

这一调用将一块 256 字节大小的数据块从缓冲区写入文件。

文件的读写：fread() 和 fwrite()

要保存一个含 10 个 double 值的数组，可以这么做

```
double arr[10];
fwrite(arr, sizeof(double), 10, fp)
```

这一调用将 arr 数组中的数据写入文件，数据分为 10 块，每块都是 double 大小。

文件的读写：fread() 和 fwrite()

fread() 从文件中读取二进制数据，其原型为

```
size_t fread(const void * ptr, size_t size, size_t
nmemb, FILE * fp);
```

其中：

- 指针 ptr 是读入文件数据的内存地址。
- size 表示要读入的数据块的大小（以字节为单位）。
- nmemb 表示数据块的数目。
- fp 指定要读入的文件
- 返回成功读入的项目数。正常情况下，它与 nmemb 相等，若有错，返回值会小于 nmemb。

文件的读写：fread() 和 fwrite()

要恢复前一例子中保存的含 10 个 double 值的数组，可以这么做

```
double arr[10];
fread(arr, sizeof(double), 10, fp)
```

这一调用将 10 个 double 值复制到 arr 数组中。

文件的读写：fread() 和 fwrite()

如果有如下的结构体类型：

```
struct student_type
{
  char name[10];
  int num;
  int age;
  char addr[30];
} stu[40];
```

结构体数组 stu 有 40 个元素，每一个元素用来存放一个学生的数据。

文件的读写：fread() 和 fwrite()

假设学生的数据已经存放在磁盘文件中，可以用下面的 for 语句和 fread() 读入 40 个学生的数据：

```
for(i=0; i<40; i++)
  fread(&stu[i], sizeof(struct student_type), 1, fp);
```

或：

```
fread(stu, sizeof(struct student_type), 40, fp);
```

文件的读写：fread() 和 fwrite()

以下程序可以将内存中的学生数据输出到磁盘文件中去：

```
for(i=0; i<40; i++)
 fwrite(&stu[i], sizeof(struct student_type), 1, fp);
```

或者只写一次

```
fwrite(stu, sizeof(struct student_type), 40, fp);
```

文件的读写：fread() 和 fwrite() 的应用举例

例　从键盘上输入一批学生数据，然后存储到磁盘上。

文件的读写：fread() 和 fwrite() 的应用举例 I

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define SIZE 4
4  void save(void);
5
6  struct stu_type
7  {
8    char name[20];
9    int num;
10   int age;
11   char addr[15];
12 } stu[SIZE];
13
```

文件的读写：fread() 和 fwrite() 的应用举例 II

```
14 int main(void)
15 {
16   int i;
17   FILE * fp;
18   printf("sizeof(structstud)=%d\n", (int) sizeof(
     struct stu_type));
19   printf("Please input the 4 student information, "
20       "including name, num, age, address\n");
21   for(i = 0; i < SIZE; i++)
22     scanf("%s%d%d%s", stu[i].name, &stu[i].num,
23           &stu[i].age, stu[i].addr);
24   save();
25   printf("\nThe information of the 4 students is:\n")
     ;
```

文件的读写：fread() 和 fwrite() 的应用举例 III

```
26    fp = fopen("stu_list.txt","rb");
27
28    fread(stu, sizeof(struct stu_type), SIZE, fp);
29    for(i = 0; i < SIZE; i++)
30    {
31  //      fread(&stu[i], sizeof(struct stu_type), 1, fp);
32        printf("%-10s%4d%4d%15s\n", stu[i].name, stu[i].
          num,
33                stu[i].age, stu[i].addr);
34    }
35    fclose(fp);
36
37    return 0;
38  }
```

文件的读写：fread() 和 fwrite() 的应用举例 IV

```
39
40 void save(void)
41 {
42   FILE * fp;
43   int i;
44
45   if ((fp = fopen("stu_list.txt", "wb")) == NULL) {
46     printf("Cannot open file!\n");
47     exit(1);
48   }
49   for (i = 0; i < SIZE; i++) {
50     if (fwrite(&stu[i], sizeof(struct stu_type), 1,
       fp) != 1)
51       printf("file write error.\n");
```

文件的读写：fread() 和 fwrite() 的应用举例 V

```
52      }
53      fclose(fp);
54
55  }
```

文件的读写：fread() 和 fwrite() 的应用举例

例　将一个浮点型数组读入磁盘文件，并从磁盘文件中读取数据显示在屏幕中。

文件的读写：fread() 和 fwrite() 的应用举例 I

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void)
4  {
5    float buffer[] = {1.0, 2.0, 3.0};
6    float read[3];
7    FILE * fp;
8
9    if ((fp = fopen("file3.txt", "wb")) == NULL) {
10     printf("Cannot open files.\n");
11     exit(0);
12   }
13   fwrite(buffer, 1, sizeof(buffer), fp);
```

文件的读写：fread() 和 fwrite() 的应用举例 II

```
14    fclose(fp);
15
16    if ((fp = fopen("file3.txt", "rb")) == NULL) {
17      printf("Cannot open files.\n");
18      exit(0);
19    }
20    fread(read, 1, sizeof(read), fp);
21    printf("%f %f %f\n", read[0], read[1], read[2]);
22    fclose(fp);
23
24    return 0;
25  }
```

4. 文件的定位

文件的定位

文件中有一个位置指针，指向当前读写的位置。每当进行一次读写后，该指针自动指向下一个字符的位置。可以用 `ftell()` 获得当前的位置指针，也可以用 `rewind()/fseek()` 改变位置指针，使其指向需要读写的位置。

文件的定位：rewind()

函数原型声明：

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

使文件 fp 的位置指针指向文件开始。

文件的定位：rewind()

例　把一个文件的内容显示在屏幕上，并同时复制到另一个文件。

文件的定位：rewind() I

```
1  // rewind.c:
2  #include <stdio.h>
3  #include <stdlib.h>
4  int main(int argc, char * argv[])
5  {
6    int ch;
7    FILE * fsrc, * fdes;
8
9    if (argc < 3) {
10     printf("Usage: %s srcfile desfile\n", argv[0]);
11     exit(1);
12   }
13
```

文件的定位：rewind() II

```
14   if ((fsrc = fopen(argv[1], "r")) == NULL ||
15       (fdes = fopen(argv[2], "w")) == NULL) {
16     printf("Cannot open files!\n");
17     exit(1);
18   }
19   while ((ch = getc(fsrc)) != EOF)
20     putchar(ch);
21
22   rewind(fsrc);
23   while ((ch = getc(fsrc)) != EOF)
24     fputc(ch, fdes);
25
26   fclose(fsrc);
27   fclose(fdes);
```

文件的定位：rewind() III

```
28
29      return 0;
30  }
```

文件的定位：rewind()

假设有文件 file.txt，其中内容为

```
C primer plus
C programming
```

则执行结果为

```
$ gcc rewind.c -o rewind
$ ./rewind file.txt file1.txt
C primer plus
C programming
```

文件的定位：fseek()

对流式文件可以进行顺序读写，也可以进行随机读写，关键在于控制文件的位置指针。用 fseek() 可以实现改变文件的位置指针。

- 函数原型声明

```
int fseek(FILE * stream, long offset, int
fromwhere);
```

- 功能：把文件的位置指针从起始点开始，移动指定位移量的字节数。
- 若一切正常，fseek() 的返回值为 0；若有错，如试图移动超出文件范围，则返回值为-1。

文件的定位：fseek()

起始点	符号常量	值
文件开始位置	SEEK_SET	0
当前位置	SEEK_CUR	1
文件尾	SEEK_END	2

文件的定位：fseek()

```
fseek(fp, 0L, SEEK_SET);   //找到文件的开始处
fseek(fp, 10L, SEEK_SET);  //找到文件的第 10 个字节
fseek(fp, 2L, SEEK_CUR);   //从文件的当前位置向前移动
                           //2 个字节
fseek(fp, 0L, SEEK_END);   //到达文件结尾处
fseek(fp, -10L, SEEK_END);//从文件结尾处退回 10 个字节
```

文件的定位：ftell()

ftell() 通过距文件开始处的字节数来确定文件的当前位置。在 ANSI C 下，该定义适用于以二进制模式打开的文件，但对以文本模式打开的文件来讲，不一定是这样。例如：

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

变量 i 存放当前位置，如调用函数时出错 (如不存在 fp 文件)，则输出"error"。

文件的定位：fseek() 和 ftell() 的应用举例

例　读取文件名，反序显示一个文件。

文件的定位：fseek() 和 ftell() 的应用举例 I

```
1  // reverse.c
2  #include<stdio.h>
3  #include<stdlib.h>
4  #define CNTL_Z '\032'
5  #define SLEN 50
6  int main(void)
7  {
8    char file[SLEN];
9    char ch;
10   FILE * fp;
11   long count, last;
12
```

文件的定位：fseek() 和 ftell() 的应用举例 II

```
13    puts("Enter the name of the file to be processed: "
      );
14    gets(file);
15
16    if ((fp = fopen(file, "rb"))==NULL) {
17      printf("reverse can't be open %s\n", file);
18      exit(1);
19    }
20
21    fseek(fp, 0L, SEEK_END);
22    last = ftell(fp);
23
24    for (count = 1L; count <= last; count++) {
25      fseek(fp, -count, SEEK_END);
```

文件的定位：fseek() 和 ftell() 的应用举例 III

```
26      ch = getc(fp);
27      if (ch != CNTL_Z && ch != '\r')
28        putchar(ch);
29    }
30
31    putchar('\n');
32    fclose(fp);
33
34    return 0;
35  }
```

文件的定位：fseek() 和 ftell() 的应用举例

```
// file4
Hello World!
I love WHU!
```

```
Enter the name of the file to be processed:
file4
!UHW evol I
!dlroW olleH
```

文件的定位：fseek() 和 ftell() 的应用举例

例　创建一个 double 型数值的文件，然后允许你访问这些内容。

文件的定位：fseek() 和 ftell() 的应用举例 I

```
1  // randbin.c
2  #include <stdio.h>
3  #include <stdlib.h>
4  #define SIZE 1000
5
6  int main(void)
7  {
8    double arr[SIZE];
9    double value;
10   const char * file = "number.dat";
11   int i;
12   long pos;
13   FILE * fp;
```

文件的定位：fseek() 和 ftell() 的应用举例 II

```
14
15    // Creat an array with double elements
16    for (i = 0; i < SIZE; i++)
17      arr[i] = 100.0 * i + 1.0 / (i + 1);
18
19    // Attempt to open file
20    if ((fp = fopen(file, "wb")) == NULL) {
21      fprintf(stderr, "Could not open %s for output",
        file);
22      exit(1);
23    }
24
25    // Write the data in the array into file with
      binary mode
```

文件的定位：fseek() 和 ftell() 的应用举例 III

```
26    fwrite(arr, sizeof(double), SIZE, fp);
27
28    // Close the file
29    fclose(fp);
30
31    // Attempt to open file
32    if ((fp = fopen(file, "rb")) == NULL) {
33      fprintf(stderr, "Could not open %s for random
        acess", file);
34      exit(1);
35    }
36
37    // Read selected item in the file
```

文件的定位：fseek() 和 ftell() 的应用举例 IV

```
38    printf("Enter an index in the range 0-%d\n", SIZE
      -1);
39    scanf("%d", &i);
40
41    while(i >= 0 && i < SIZE) {
42      pos = (long) i * sizeof(double); // compute
        offset
43      fseek(fp, pos, SEEK_SET);
44      fread(&value, sizeof(double), 1, fp);
45      printf("The value there is %f.\n", value);
46      printf("Next index (out of range to quit):\n");
47      scanf("%d", &i);
48    }
49    fclose(fp);
```

文件的定位：fseek() 和 ftell() 的应用举例 V

```
50    puts("Bye!");
51
52    return 0;
53 }
```

文件的定位：fseek() 和 ftell() 的应用举例 I

```
Enter an index in the range 0-999
1
The value there is 100.500000.
Next index (out of range to quit):
4
The value there is 400.200000.
Next index (out of range to quit):
5
The value there is 500.166667.
Next index (out of range to quit):
100
The value there is 10000.009901.
Next index (out of range to quit):
```

文件的定位：fseek() 和 ftell() 的应用举例 II

```
-1
Bye!
```

5. 出错检测

出错检测

C 标准提供一些函数用来检查输入输出函数调用中的错误。

出错检测：ferror()

在文件操作时，如果出错，除了操作函数的返回值有所反应外（如 fopen() 返回 NULL），还可以用 ferror() 获得是否出错。

- 函数原型声明：

```
int ferror(FILE * stream);
```

- 功能：若上一次文件操作未出错，返回 0；否则返回非 0。

出错检测：cleanerr 函数

- 函数原型声明

```
int ferror(FILE * stream);
```

- 功能：使文件错误标志和文件结束标志置为 0。
- 文件操作出现错误后，`ferror(fp)` 函数值为一个非 `0` 值，该错误信息将一直保留在系统中，在调用 `clearerr(fp)` 函数后，`ferror(fp)` 函数值变成 `0`。