C/C++

文件 IO

张晓平

武汉大学数学与统计学院

Table of contents

文件类型

文件类型

文件

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，其存储形式是：

```
00100111  00010000
```

- 在二进制文件中仍以上述方式存储；
- 在文本文件中，存储形式为

  ```
  31H，30H，30H，30H，30H
  ```

  占五个字节，分别是字符 1，0，0，0，0 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

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。

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

对于缓冲文件系统，

- 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。

文件类型

对于缓冲文件系统，

- 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。

注

- 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

FILE

在缓冲文件系统中，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，如

- 文件名
- 文件当前的读写位置
- 文件操作方式
- …

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

文件指针

```
FILE * fp;
```

注

只有通过文件指针，才能使用相应的文件。

文件的打开与关闭

文件的打开与关闭

文件操作的过程

先打开，后读写，最后关闭。

文件的打开与关闭

文件的打开

ANSI C 用函数 fopen() 打开文件。

- 函数原型声明

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

- 参数说明
  - path 为字符串，包含欲打开的文件路径及文件名；
  - mode 为字符串，表示“文件打开模式”。

文件的打开与关闭

例

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

表示要打开名字为 `file1.txt` 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.txt` 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 1: 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 2: 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

- "r" 模式

  只能从文件中读取数据。该文件必须已经存在，否在打开失败。

- "w" 模式

  只能向文件写数据。若指定文件不存在则创建它；若存在则先删除它再重建一个新文件。

- "a" 模式

  向文件添加新数据（不删除原数据）。要求指定文件已经存在，否则打开失败，打开时位置指针移到文件末尾。

文件的打开与关闭

- "r+" 模式

  可读/写数据，该文件必须已经存在，否在打开失败。

- "w+" 模式

  可读/写数据。用该模式新建一个文件，先向该文件写数据，然后可读取其中的数据。

- "a+" 模式

  可读/写数据，　原来的文件不被删除，位置指针移到文件末尾。

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：文件的关闭（fclose）

文件的关闭

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该文件进行读写操作。可用 `fclose()` 关闭文件。

函数`fclose()`

- 函数原型声明

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

- 返回一个 `int` 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：文件的关闭（fclose）

注

执行完文件操作后，要进行“关闭文件”操作。 虽然程序在结束前会自动关闭所有的打开文件，但文件打开过多会导致系统运行缓慢，这时就要自行手动关闭不再使用的文件，来提高系统整体的执行效率。

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

```
// count.c -- using standard I/O
#include <stdio.h>
#include <stdlib.h> // ANSI C exit() prototype
int main(int argc, char *argv[])
{
  int ch;
  FILE * fp;
  long count = 0;
  if (argc != 2) {
    printf("Usage: %s filename\n", argv[0]);
    exit(1);
  }
  if ((fp = fopen(argv[1], "r")) == NULL) {
    printf("Can't open %s\n", argv[1]);
    exit(1);
  }
```

文件的打开与关闭 ii

```
  while ((ch = getc(fp)) != EOF) {
    putc(ch, stdout); // same as putchar(ch);
    count++;
  }
  fclose(fp);
  printf("\nFile %s has %ld characters.\n", argv[1],
  count);
  return 0;
}
```

文件的读写

文件的读写

文本文件的读写

文本文件的读写

- **字符存取函数**：fputc() 和 fgetc()
- **字符串存取函数**：fputs() 和 fgets()
- **格式化存取函数**：fprintf() 和 fscanf()

文本文件的读写：fputc()

函数原型

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

函数说明

- 将 c 转为 unsigned char 后写入 stream 指定的文件中。
- 成功时返回字符 c 的 ASCII 码，失败时返回 EOF。

文本文件的读写：fgetc()

函数原型

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

函数说明

- 从 stream 指定的文件中读取一个字符，然后将光标移动到下一个字符。
- 若字符读取成功，则返回所读取的字符，否则返回 EOF。
- 要判断文件是否读取完毕，可用 feof() 进行检查。未完返回 0，已完返回非零值。

文本文件的读写：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。

文本文件的读写：fgetc() i

```
// file2screen.c:
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char * argv[])
{
  FILE * fp;
  int ch;
  if (argc != 2) {
    printf("Usage: %s filename\n", argv[0]);
    exit(1);
  }
  if ((fp = fopen(argv[1], "r")) == NULL) {
    printf("Can't open %s\n", argv[1]);
    exit(1);
  }
  while ((ch = fgetc(fp)) != EOF) {
```

文本文件的读写：fgetc() ii

```
    fputc(ch, stdout); // same as putchar(ch);
  }
  return 0;
}
```

文本文件的读写：fgetc() i

```
// fgetc1.c:
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char * argv[])
{
  FILE * fp;
  int ch;
  if (argc != 2) {
    printf("Usage: %s filename\n", argv[0]);
    exit(1);
  }
  if ((fp = fopen(argv[1], "r")) == NULL) {
    printf("Can't open %s\n", argv[1]);
    exit(1);
  }
  while ( !feof(fp) )
```

文本文件的读写：fgetc() ii

```
    putchar(fgetc(fp));
  fclose(fp);
  return 0;
}
```

文本文件的读写：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 EOF 为止。

文本文件的读写：fputc() 和 fgetc() 的应用举例

```c
// screen2file.c:
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  char ch;
  FILE *fp;
  if ( (fp = fopen("file1.txt", "w")) == NULL) {
    printf("Cannot open file1.txt!\n");
    exit(1);
  }
  while ((ch = getchar()) != EOF)
    fputc(ch, fp);
  fclose(fp);
  return 0;
}
```

文本文件的读写：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。

文本文件的读写：fputc() 和 fgetc() 的应用举例 i

```
// copy.c:
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char * argv[])
{
  char ch;
  FILE * fsrc, * fdes;
  if (argc < 3) {
    printf("Usage: %s srcfile desfile\n", argv[0]);
    exit(1);
  }
  if ((fsrc = fopen(argv[1], "r")) == NULL ||
      (fdes = fopen(argv[2], "w")) == NULL) {
    printf("Cannot open files!\n");
    exit(1);
  }
```

文本文件的读写：fputc() 和 fgetc() 的应用举例 ii

```
  while ((ch = getc(fsrc)) != EOF)
    fputc(ch, fdes);
  printf("Successed copy srcfile to desfile\n");
  fclose(fsrc);
  fclose(fdes);
  return 0;
}
```

文本文件的读写：fputss() 和 fgets()

```
int fputs(const char *, FILE *);
```

向指定文件输出一个字符串，如

```
fputs("Hello world", fp);
```

把字符串 "Hello world" 输出到 fp 指定的文件。fputs() 的第一个参数可以是字符串常量、字符数组名或字符型指针。若输出成功，则返回 0，否则返回 EOF。

文本文件的读写：fputss() 和 fgets()

```
char * fgets(char *, FILE *);
```

从指定文件中读入一个字符串，如

```
fgets(str, n, fp)
```

n 为要求得到的字符个数，但只从 fp 指定的文件中读入 n-1 个字符，然后在最后加一个 '\0'，因此得到的字符串共有 n 个字符，把它们存放在字符数组 str 中。如果在读完 n-1 个字符之前遇到换行符或 EOF，读入结束。

文本文件的读写：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，
- printf() 和 scanf() 的读写对象是终端。

文本文件的读写：fscanf() 和 fprintf()

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。

文本文件的读写：fscanf() 和 fprintf()

例

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

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。

若 i=3, t=4.5，则输出到磁盘文件上的是以下字符串：

```
3, 4.50
```

文本文件的读写：fscanf() 和 fprintf()

例

可用 fscanf() 从文件上读入 ASCII 字符：

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

若文件中有以下字符：

```
3, 4.5
```

则将文件的数据 3 送给变量 i，4.5 送给变量 t。

文本文件的读写：fscanf() 和 fprintf() 的应用举例

例

编制程序，向文件添加单词，并显示文件内容在屏幕上。

文本文件的读写：fscanf() 和 fprintf() 的应用举例 i

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 40
int main(int argc, char * argv[])
{
  FILE * fp;
  char words[MAX];
  if (argc != 2) {
    printf("Usage: %s filename\n", argv[0]);
    exit(1);
  }
  if((fp = fopen(argv[1], "a+")) == NULL) {
    fprintf(stdout, "Can't open \"words\" file.\n");
    exit(1);
  }
  puts("Enter words to add to the file ");
```

文本文件的读写：fscanf() 和 fprintf() 的应用举例 ii

```
  puts("(press the Enter key at the beginning of ");
  puts("a line to terminate.)");
  while ( gets(words) != NULL && words[0] != '\0')
    fprintf(fp, "%s", words);
  puts("File contents:");
  rewind(fp);
  while(fscanf(fp, "%s", words)==1)
    puts(words);
  fclose(fp);
  return 0;
}
```

文本文件的读写：fscanf() 和 fprintf()

用 fprintf() 和 fscanf() 进行文件操作时，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

二进制文件的读写

二进制文件的读写：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，也会导致存储内容的不同。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。

二进制文件的读写：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。

二进制文件的读写：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：

- ptr 指定要写入的数据块的地址。
- size 指定要写入的数据块的大小（以字节为单位）。
- nmemb 指定数据块的数目。
- 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。

二进制文件的读写：fread() 和 fwrite()

例

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。

二进制文件的读写：fread() 和 fwrite()

例

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。

二进制文件的读写：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：

- ptr 指定读入文件数据的内存地址。
- size 指定要读入的数据块的大小（以字节为单位）。
- nmemb 指定数据块的数目。
- 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。

二进制文件的读写：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。

二进制文件的读写：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。

二进制文件的读写：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);
```

二进制文件的读写：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)
;
```

二进制文件的读写：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。

二进制文件的读写：fread() 和 fwrite() 的应用举例 i

```c
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  float buffer[] = {1.0, 2.0, 3.0};
  float read[3];
  FILE * fp;
  if ((fp = fopen("file3.txt", "wb")) == NULL) {
    printf("Cannot open files.\n");
    exit(0);
  }
  fwrite(buffer, sizeof(buffer), 1, fp);
  fclose(fp);
  if ((fp = fopen("file3.txt", "rb")) == NULL) {
    printf("Cannot open files.\n");
    exit(0);
```

二进制文件的读写：fread() 和 fwrite() 的应用举例 ii

```
    }
    fread(read, sizeof(read), 1, fp);
    printf("%f %f %f\n", read[0], read[1], read[2]);
    fclose(fp);
    return 0;
}
```

二进制文件的读写：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。

二进制文件的读写：fread() 和 fwrite() 的应用举例 i

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 4
void save(void);
struct stu_type
{
  char name[20];
  int num;
  int age;
  char addr[15];
} stu[SIZE];

int main(void)
{
  int i;
  FILE * fp;
```

二进制文件的读写：fread() 和 fwrite() 的应用举例 ii

```
printf("sizeof(structstud)=%d\n", (int) sizeof(
struct stu_type));
printf("Please input the 4 student information, "
   "including name, num, age, address\n");
for(i = 0; i < SIZE; i++)
  scanf("%s%d%d%s", stu[i].name, &stu[i].num,
        &stu[i].age, stu[i].addr);
save();
printf("\nThe information of the 4 students is:\n");
fp = fopen("stu_list.txt", "rb");
fread(stu, sizeof(struct stu_type), SIZE, fp);
for(i = 0; i < SIZE; i++) {
  printf("%-10s%4d%4d%15s\n", stu[i].name, stu[i].
  num,
         stu[i].age, stu[i].addr);
}
```

二进制文件的读写：fread() 和 fwrite() 的应用举例 iii

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

二进制文件的读写：fread() 和 fwrite() 的应用举例 iv

```
  fclose(fp);
}
```

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()

```
// rewind.c:
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char * argv[])
{
  int ch;
  FILE * fsrc, * fdes;
  if (argc < 3) {
    printf("Usage: %s srcfile desfile\n", argv[0]);
    exit(1);
  }
  if ((fsrc = fopen(argv[1], "r")) == NULL ||
      (fdes = fopen(argv[2], "w")) == NULL) {
    printf("Cannot open files!\n");
    exit(1);
  }
```

文件的定位：rewind() ii

```
  while ((ch = getc(fsrc)) != EOF)
    putchar(ch);
  rewind(fsrc);
  while ((ch = getc(fsrc)) != EOF)
    fputc(ch, fdes);
  fclose(fsrc);
  fclose(fdes);
  return 0;
}
```

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()

- 原型声明

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

- 功能：返回 stream 指定的文件的当前位置
- 返回位置标识符的当前值。若发生错误，则返回 -1L。

例

```
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

```
// reverse.c
#include<stdio.h>
#include<stdlib.h>
#define CNTL_Z '\032'
#define SLEN 50
int main(void)
{
  char file[SLEN];
  char ch;
  FILE * fp;
  long count, last;
  puts("Enter the name of the file to be processed: ")
  ;
  gets(file);
  if ((fp = fopen(file, "rb"))==NULL) {
    printf("reverse can't be open %s\n", file);
```

文件的定位：fseek() 和 ftell() 的应用举例 ii

```
    exit(1);
  }
  fseek(fp, 0L, SEEK_END);
  last = ftell(fp);
  for (count = 1L; count <= last; count++) {
    fseek(fp, -count, SEEK_END);
    ch = getc(fp);
    if (ch != CNTL_Z && ch != '\r')
      putchar(ch);
  }
  putchar('\n');
  fclose(fp);
  return 0;
}
```

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

```c
// randbin.c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 1000
int main(void)
{
  double arr[SIZE];
  double value;
  const char * file = "number.dat";
  int i;
  long pos;
  FILE * fp;
  // Creat an array with double elements
  for (i = 0; i < SIZE; i++)
    arr[i] = 100.0 * i + 1.0 / (i + 1);
  // Attempt to open file
```

文件的定位：fseek() 和 ftell() 的应用举例 ii

```
if ((fp = fopen(file, "wb")) == NULL) {
  fprintf(stderr, "Could not open %s for output",
  file);
  exit(1);
}
// Write the data in the array into file with binary
 mode
fwrite(arr, sizeof(double), SIZE, fp);
// Close the file
fclose(fp);
// Attempt to open file
if ((fp = fopen(file, "rb")) == NULL) {
  fprintf(stderr, "Could not open %s for random
  acess", file);
  exit(1);
}
```

文件的定位：fseek() 和 ftell() 的应用举例 iii

```
  // Read selected item in the file
  printf("Enter an index in the range 0-%d\n", SIZE-1)
  ;
  while( (scanf("%d", &i) == 1) && (i >= 0 && i < SIZE
  ) ) {
    pos = (long) i * sizeof(double); // compute offset
    fseek(fp, pos, SEEK_SET);
    fread(&value, sizeof(double), 1, fp);
    printf("The value there is %f.\n", value);
    printf("Next index (out of range to quit):\n");
  }
  fclose(fp);
  puts("Bye!");
  return 0;
}
```

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):
-1
Bye!
```