13 文件输入/输出

内容提要

- 和文件进行通信
- 标准I/O

1 与文件进行通信

1 与文件进行通信

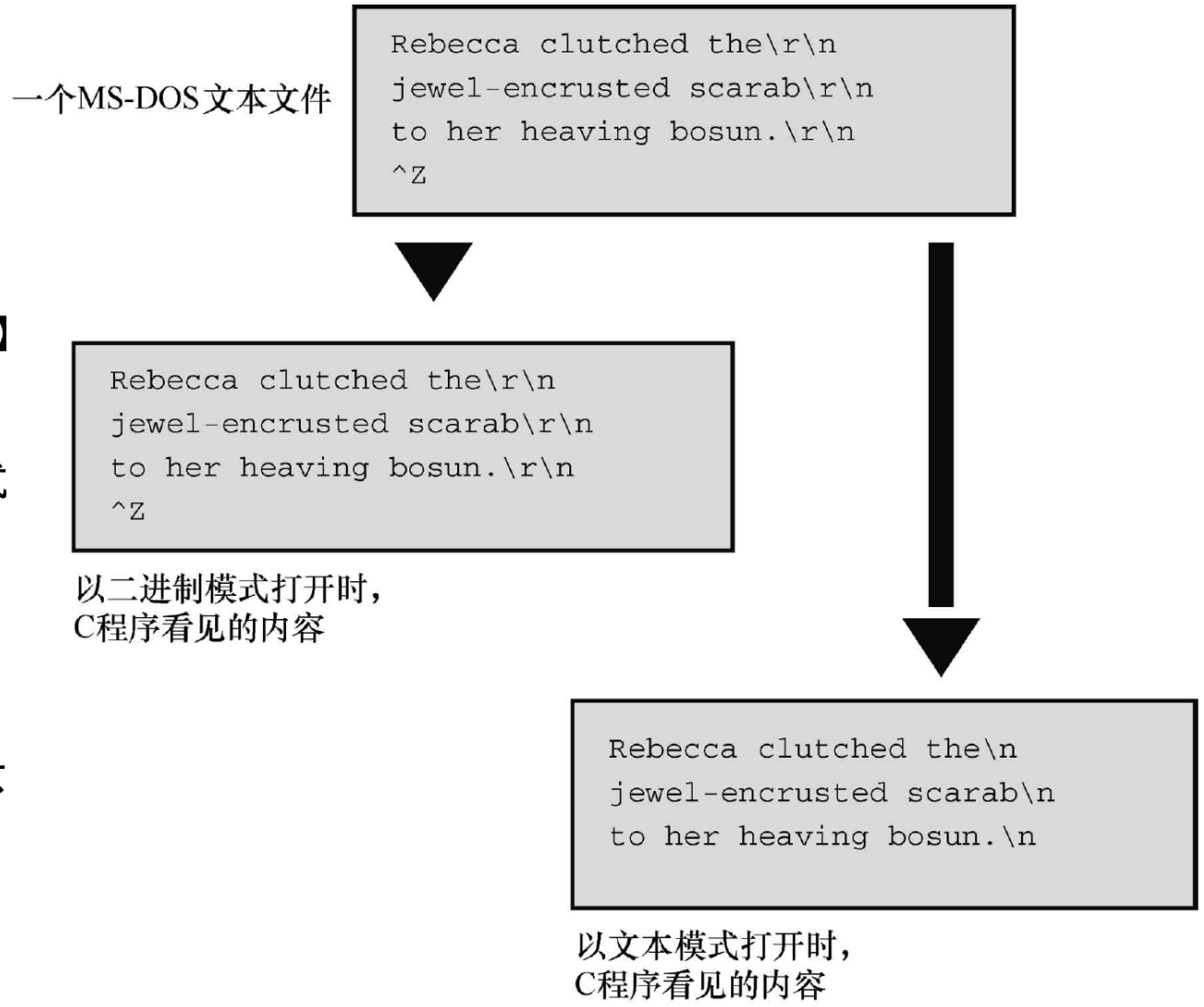
- 一般性的需求
  - 程序从文件中读取信息(Input)
  - 程序将信息写入文件(Output)
- I/O(Input/Output)
- 使用特殊的I/O函数读取文件中的信息或把信息写入文件

1.1 文件是什么

- 文件（file），通常是在磁盘或硬盘上的一段已命名的存储区
  - 存储设备上的一段区域，存储了某些信息
  - 用文件名来标识
- 用户角度
  - 如，stdio.h为文件的名称，该文件中包含一些有用的信息
- 系统角度
  - 操作系统。如，大型文件会被分开储存，或者包含一些额外的数据，方便操作系统确定文件的种类
- 程序员角度
  - C把文件看作是一系列连续的字节，每个字节都能被单独读取
- C提供两种文件模式
  - 文本模式，二进制模式

1.2 文本模式和二进制模式

- 文本文件和二进制文件
  - 文件内容以二进制形式储存
  - 文本文件
    - 使用二进制编码的字符（如 ASCII或Unicode）
    - 如，整数123，表示成ASCII的”123”【人可以读出】
  - 二进制文件
    - 机器语言代码或数值数据（和内存相同的表达方式
    - 如，整数8，表示成0X0008；或图片或音乐编码
- C提供两种访问文件的途径
  - 二进制视：程序访问文件的每个字节
  - 文本视图：程序看到的内容可能和文件的内容不同，和操作系统有关系
- 文本文件可以文本/二进制模式读写



示例

- 文本文件是特殊的二进制文件，可以处理成用户直接看懂的方式
  - 文本文件，二进制和文本模式打开（数据变长，数据的每一项是通过字符串表示的）
  - 二进制文件，二进制模式打开（数据每一项用内存的二进制方式表达，因而是固定长度）

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded Text
00000000	48	65	6C	6C	6F	0D	0A	31	32	33	0D	0A	31	32	33	2E	H e l l o . . 1 2 3 . . 1 2 3 .
00000010	34	35															4 5

```
1   Hello
2   123
3   123.45
```

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded Text
00000000	48	65	6C	6C	6F	00	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	H e l l o . . . . .
00000010	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	. . . . .
00000020	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	. . . . .
00000030	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	. . . . .
00000040	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	. . . . .
00000050	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	. . . . .
00000060	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	. . . . .
00000070	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	. . . . .
00000080	7B	00	00	00	CD	CC	CC	CC	CC	DC	5E	40					{ . . . . . ^ @

1.3 I/O级别

- I/O级别（即处理文件访问的两个级别）
  - 底层I/O(low-level I/O)：使用操作系统提供的基本I/O服务
  - 标准高级I/O(standard high-levelI/O)：使用一个标准的C库函数包和stdio.h头文件中的定义
- ANSI C只支持标准I/O包
  - 因为无法保证所有的操作系统都使用相同的底层I/O 模

1.4 标准文件

- C程序自动打开三个文件
  - 标准输入。默认标准输入通常是键盘
  - 标准输出。默认标准输出为显示器
  - 标准错误。默认标准错误输出为显示器
- 标准输入
  - 用getchar(), gets()和scanf()读取的文件提供输入
- 标准输出
  - putchar(), puts(), printf()是程序输出对象
- 标准错误输出
  - 标准错误输出提供了一个逻辑上不同的地方来发送错误消息

2 标准I/O

2 标准I/O

- 标准I/O对比低级I/O的优势：
  - 标准I/O有许多专门的函数简化处理不同I/O的问题
  - 对输入输出都是缓冲的。提高数据传输率

程序清单13.1 count.c

- 检查argc的值，查看是否有命令行参数
  - 如果没有，打印一条消息并退出程序。字符串argv[0]是该程序的名称。错误消息的描述会随可执行文件名的改变而自动改变
- exit()函数关闭所有打开的文件并结束程序
  - 通常的惯例是：正常结束的程序传递0，异常结束的程序传递非零值

```
1.  int main(int argc, char *argv[]){
2.      int ch;  // place to store each character as read
3.      FILE *fp; // "file pointer"
4.      unsigned long count = 0;
5.      if (argc != 2){
6.          printf("Usage: %s filename\n", argv[0]);
7.          exit(EXIT_FAILURE);
8.      }
9.      if ((fp = fopen(argv[1], "r")) == NULL){
10.         printf("Can't open %s\n", argv[1]);
11.         exit(EXIT_FAILURE);
12.     }
13.     while ((ch = getc(fp)) != EOF){
14.         putc(ch,stdout);  // same as putchar(ch);
15.         count++;
16.     }
17.     fclose(fp);
18.     printf("%s has %lu characters\n", argv[1], count);
19.     return 0;
20. }
```

2.1 检查命令行参数

- 一些操作系统可能不识别argv[0]，所以这种用法并非完全可移植
- 根据ANSI C的规定，在最初调用的main()中使用return与调用exit 0的效果相同
- 如果main()在一个递归程序中，exit()仍然会终止程序,但是return只会把控制权交给上一级递归，直至最初的一级。然后return结束程序
- return和exit()的另一个区别是，即使在其他函数中（除main()以外）调用exit()也能结束整个程序

2.2 fopen( )函数

- fp = fopen(argv[1], "r")
  - 参数1，包含该文件名的字符串的地址；
  - 参数2，用于指定文件打开模式的一个字符串
  - 返回值fp
    - 打开失败，返回空
    - 成功打开文件，返回文件指针（file pointer），其他I/O函数可以使用这个指针指定该文件
      - 文件指针的类型是指向FILE的指针。FILE是一个定义在stdio.h中的派生类型。
      - 文件指针fp并不指向实际的文件，它指向一个包含文件信息的数据对象，其中包含操作文件的I/O函数所用的缓冲区信息
- 模式字符串
  - 二进制文件：带b字母的模式
  - 读写时
    - 带+字母，则表明以更新模式打开文件（可以读和写）
    - r：读模式；w：写模式；a：追加。

2.2 fopen( )函数

- fopen( )函数

表 13.1 fopen()的模式字符串

模式字符串	含义
"r"	以读模式打开文件
"w"	以写模式打开文件，把现有文件的长度截为 0，如果文件不存在，则创建一个新文件
"a"	以写模式打开文件，在现有文件末尾添加内容，如果文件不存在，则创建一个新文件
"r+"	以更新模式打开文件（即可以读写文件）
"w+"	以更新模式打开文件（即，读和写），如果文件存在，则将其长度截为 0；如果文件不存在，则创建一个新文件
"a+"	以更新模式打开文件（即，读和写），在现有文件的末尾添加内容，如果文件不存在则创建一个新文件；可以读整个文件，但是只能从末尾添加内容

模式字符串	含义
"rb"、"wb"、"ab"、"rb+"、"r+b"、"wb+"、"w+b"、"ab+"、"a+b"	与上一个模式类似，但是以二进制模式而不是文本模式打开文件
"wx"、"wbx"、"w+x"、"wb+x"或"w+bx"	（C11）类似非 x 模式，但是如果文件已存在或以独占模式打开文件，则打开文件失败

2.3 getc( )函数和putc( )函数

- getc( )函数和putc( )函数
  - ch = getchar( ); //从标准输入中获取一个字符
  - ch = getc(fp); //从指针fp指定的文件获得一个字符
  - putc(ch, fpout); //将字符写入到指针fpout指定的文件中
    - putc(ch, stdout)等价于putchar(ch)

2.4 文件结尾

- 从文件中读取数据的程序在读到文件结尾时要停止
- 判断文件结尾
  - 如果在尝试读入字符时发现已经到达文件结尾，getc()函数返回一个特殊值EOF。
  - 对文件输入使用入口条件循环，避免试图读取空文件

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

2.5 fclose()函数

- fclose()函数
  - fclose( )函数关闭由指针fp指定的文件
    - 文件成功关闭，fclose返回0；
    - 否则，返回EOF
    - 磁盘已满或磁盘被移走或I/O错误等，会导致fclose()函数执行失败。

2.6 指向标准文件的指针

- 标准文件指针
  - 标准文件 文件指针 使用设备
  - 标准输入 stdin 键盘
  - 标准输出 stdout 显示器
  - 标准错误 stderr 显示器

3 一个简单的文件压缩程序

3 一个简单的文件压缩程序

13.2 reducto.c程序

把一个文件中选定的数据拷贝到另一个文件中。该程序同时打开了两个文件，以"r"模式打开一个，以"w"模式打开另一个

```
1.  #define LEN 40
2.  int main(int argc, char *argv[]){
3.      FILE  *in, *out;   // declare two FILE pointers
4.      char name[LEN];  // storage for output filename
5.      int count = 0;
6.      // check for command-line arguments
7.      if (argc < 2) {
8.          fprintf(stderr, “Usage: %s …\n", argv[0]);
9.          exit(EXIT_FAILURE);
10.     }
11.     // set up input
12.     if ((in = fopen(argv[1], "r")) == NULL)    {
13.         fprintf(stderr, “I couldn’t ...file
    \"%s\"\n", argv[1]);
14.         exit(EXIT_FAILURE);
15.     }
16.     // set up output
17.     strncpy(name,argv[1], LEN - 5); // copy filename
18.     name[LEN - 5] = '\0';
19.     strcat(name,".red"); // append .red
20.     if ((out = fopen(name, "w")) == NULL)
21.     { // open file for writing
22.         fprintf(stderr,"Can't create output.\n");
23.         exit(3);
24.     }
25.     // copy data
26.     while ((int ch = getc(in)) != EOF)
27.         if (count++ % 3 == 0)
28.             putc(ch, out); //print every 3rd char
29.     // clean up
30.     if (fclose(in) != 0 || fclose(out) != 0)
31.         fprintf(stderr,"Error in closing files\n");
32.     return 0;
33. }
```

文件I/O：fprintf()、fscanf()、fgets()和fputs()

4 fprintf(),fscanf(),fgets()和fputs()函数

- 文件I/O函数要用FILE指针指定待处理的文件
  - 与getc()、putc()类似，这些函数都要求用指向FILE的指针（如，stdout）指定一个文件，或者使用fopen()的返回值
- 13.3 addaword.c程序
- 该程序可以在文件中添加单词。使用" a+ " 模式，程序可以对文件进行读写操作
- rewind（ ）函数让程序回到文件开始处，方便while 循环打印整个文件的内容

4.1 fprintf(),fscanf()

```
1.  // 13.3 addaword.c程序
2.  /* addaword.c -- uses fprintf(), fscanf(), and
    rewind() */
3.  #define MAX 41

4.  int main(void){
5.      FILE *fp;
6.      char words[MAX];
7.      if ((fp = fopen("wordy", "a+")) == NULL){
8.          fprintf(stdout,"Can't open \"wordy\"
    file.\n");
9.          exit(EXIT_FAILURE);
10.     }
11.     puts("Enter words …; press #");
12.     puts("key at … terminate.");
13.     while ((fscanf(stdin,"%40s", words) == 1)  &&
    (words[0] != '#'))
14.         fprintf(fp, "%s\n", words);
15.
16.     puts("File contents:");
17.     rewind(fp);/* go back to beginning of file */
18.     while (fscanf(fp,"%s",words) == 1)
19.         puts(words);
20.     puts("Done!");
21.     if (fclose(fp) != 0)
22.         fprintf(stderr,"Error closing file\n");
23.
24.     return 0;
25. }
```

程序可以在文件中添加单词。使用"a+"模式，程序对文件进行读写操作

首次使用该程序，它将创建wordy文件，以便把单词存入其中。随后再使用该程序，可以在wordy文件后面添加单词。虽然"a+"模式只允许在文件末尾添加内容，但是该模式下可以读整个文件。rewind()函数让程序回到文件开始处，方便while循环打印整个文件的内容。

4.2 fgets()和fputs()函数

- fgets(buf, STLEN, fp)
  - 函数接受三个参数，第一个参数用于储存输入的地址（char *类型），第二个参数是表示输入字符串的整数，第三个参数是指向要读取的文件的文件指针
  - 字符串的最大长度代表字符的最大数目加上空字符。
- fgets函数和gets函数的区别
  - fgets在达到字符最大数目之前读完了一整行，在空字符之前加一个换行符
  - gets读取换行符后之后将其丢弃。
- fputs(buf, fp)
  - 函数接受两个参数，一个是字符串地址另一个是文件指针。fputs( )函数打印的时候不添加换行符，这点和puts()函数不同。

5 随机存储：fseek()和ftell()函数

5 随机存储：fseek()和ftell()函数

程序清单13.4 reverse.c

```
/* reverse.c -- displays a file in reverse order */
#define CNTL_Z '\032' //eof marker in DOS text files
#define SLEN 81
int main(void){
    char file[SLEN];
    char ch;
    FILE *fp;
    long count, last;

    puts("Enter the name of the file:");
    scanf("%80s", file);
    if ((fp = fopen(file,"rb")) == NULL)
    { /* read-only mode   */
        printf("reverse can't open %s\n", file);
        exit(EXIT_FAILURE);
    }

    fseek(fp, 0L, SEEK_END);  /* go to end of file */
    last = ftell(fp);
    for (count = 1L; count <= last; count++)
    {
        fseek(fp, -count, SEEK_END); //go backward
          ch = getc(fp);
          if (ch != CNTL_Z && ch != '\r') /MS-DOS
            putchar(ch);
    }
    putchar('\n');
    fclose(fp);

    return 0;
}
```

5.1 fseek()和ftell()的工作原理

- fseek( )函数
  - 第一个参数是指向被搜索文件的FILE指针
  - 第二个参数是long类型的偏移量，表示从起始点开始要移动的距离
  - 第三个参数是模式，用来表示起始点
    - 模　　式　　偏移量的起始点
    - SEEK_SET　　文件开始
    - SEEK_CUR　　当前位置
    - SEEK_END　　文件结尾
  - 返回值：一切正常，函数返回值为0；有错误出现，返回值为-1
- 示例
  - fseek(fp, 10L, SEEK_SET);　//找到文件的第10个字节
  - fseek(fp, 10L, SEEK_CUR);　//在文件的当前位置向前移动10个字节
  - fseek(fp, -10L, SEEK_END);　//从文件结尾处退回10个字节

5.1 fseek()和ftell()的工作原理

- ftell( )函数
  - ftell( )函数为long类型，它返回文件的当前位置。ftell( )函数在stdio.h被声明，函数通过返回距文件开始处的字节数目来确定文件的位置。
  - 函数在文本模式和二进制模式下的工作方式不同。文本模式下函数ftell( )返回一个可以用作fseek()第二个参数的值。

5.2 二进制模式和文本模式

- UNIX只有一种文件格式，所以不需要进行特殊的转换
- 许多MS-DOS编辑器都用Ctrl+Z标记文本文件的结尾
- 二进制模式和文本模式的另一个不同之处是：
  - MS-DOS用\r\n组合表示文本文件换行
  - 以文本模式打开相同的文件时，C程序把\r\n“看成”\n
  - 以二进制模式打开该文件时，程序能看见这两个字符

5.3 可移植性

- 理论上，fseek()和ftell()应该符合UNIX模型。但是，不同系统存在着差异，有时确实无法做到与UNIX模型一致
  - 在二进制模式中，实现不必支持SEEK_END模式
  - 在文本模式中，只有以下调用能保证其相应的行为

函数调用	效果
fseek(file, 0L, SEEK_SET)	定位至文件开始处
fseek(file, 0L, SEEK_CUR)	保持当前位置不动
fseek(file, 0L, SEEK_END)	定位至文件结尾
fseek(file,ftell-pos, SEEK_SET)	到距文件开始处 ftell-pos 的位置，ftell-pos 是 ftell()的返回值

5.4 fgetpos()和fsetpos()函数

- 用来处理较大文件的新的定位函数
  - 使用fpos_t(代表file position type,文件定义类型)代表位置
- fpos_t不是基本类型，是通过其他类型定义的，一个fpos_t类型的变量或数据对象可以指定文件中的位置，注意不能是数组类型

- fgetpos( )的函数原型
  - int fgetpos(FILE *restrict stream, fpos_t * restrict pos);
  - fsetpos( )的函数原型:
  - int fgetpos(FILE *stream, const fpos_t *pos);//
  - fpos_t的值通过调用fgetpos( )函数获取的

6 标准I/O机理

6 标准I/O机理

- 工作原理（考虑文件输入）
  - Step1 用fopen打开一个文件，（自动打开3 种标准文件），自动建立一个缓冲区，创建了一个包含文件和缓冲区相关数据的数据结构（FILE），返回一个指向该结构的指针（FILE *）。假设把该指针赋给一个指针变量fp，我们说fopen( ) 函数“打开一个流”
    - 如果以文本模式打开该文件，就获得一个文本流；
    - 如果以二进制模式打开该文件，就获得一个二进制流
  - Step2 调用stdio.h头文件声明某个输入函数， fscanf( ) 、getc( ) 或fgets( )
    - 这些输入函数会把数据复制到缓冲区
    - 数据结构和缓冲区初始化后，输入函数将读取请求的数据，文件指示器在最后被读取的字符的位置
    - 输入函数检测到已经读取的全部的字符，就会请求系统复制下一块缓冲区大小的数据到缓冲区，输入函数就可以读入全部内容。若读到所以数据的最后一个字符，会将文件结尾指示器的值设为真，下一个被调用的输入函数返回EOF

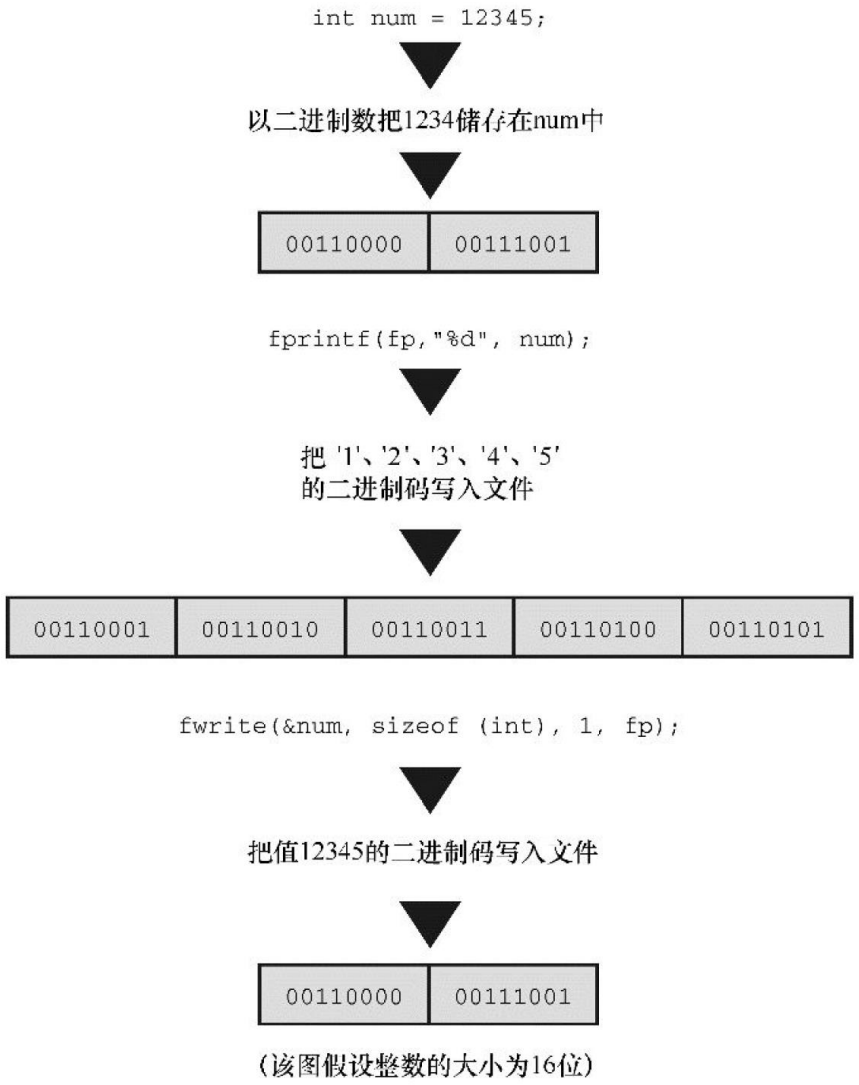
7 其他标准I/O函数

7 其他标准I/O函数

- int ungetc(int c, FILE *fp)函数
  - 将C指定的字符放回输入流中，并且下次调用标准输入函数就会读入指定的字符
  - 例如，假设要读取下一个冒号之前的所有字符，但是不包括冒号本身，可以使用getchar( ) 或getc( ) 函数读取字符到冒号，然后使用ungetc( ) 函数把冒号放回输入流中
- int fflush(FILE *fp)
  - 将缓冲区任何未写的数据发送到一个由fp指定的输出文件中去，这个过程叫刷新缓冲区
  - 若fp为空，则刷新掉所有缓冲数据。不能对输入流使用这个函数
- int setvbuf(FILE *restrict fp, char *restrict buf , int mode, size_t size)
  - 建立标准的替换缓冲区，指针 fp指定流，buf 指向将使用的存储区，若buf的值不是NULL必须创建缓冲区。size变量指定数组大小，mode在_IOFBF（完全缓冲）_IOLBF(行缓冲）_IONBF(无缓冲)中选择，成功执行返回0值

fread( )和fwrite( )函数

- 二进制I/O：fread( )和fwrite( )函数
  - 二进制形式存储：把数据存储在一个使用与程序内存具有相同表示方法的文件中
- size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *fp)
  - 第一个参数类型不固定，指针ptr要写入数据块的地址
  - size表示要写入数据块大小
  - nmemb表示写入数据块的数目
  - 例如
    - char buffer[256]; fwrite(buffer, 256,1,fp); //将一块256字节的数据块写入到文件，保存数据对象
    - char earnings [10]; fwrite(earnings, sizeof(double)10,fp); //保存数组
- size_t fread (void * ptr, size_t size, size_t nmemb, FILE * fp)
  - double earnings [10];
  - fread(earnings, sizeof(double), 10, fp);



feof与ferror

- int feof(FILE *lp)和int ferror(FILE *fp)函数
  - 用这两个函数区分返回EOF是文件结尾还是传输错误
  - 文件结尾，feof()返回非零值
  - 读写错误，ferror()回非零值

7.8 一个程序示例

程序清单13.5 append.c

```c
#define BUFSIZE 4096
#define SLEN 81
void append(FILE *source, FILE *dest);
char * s_gets(char * st, int n);
int main(void){
    FILE *fa, *fs;  // fa for append file, fs for source file
    int files = 0;  // number of files appended
    char file_app[SLEN];  // name of append file
    char file_src[SLEN];  // name of source file
    int ch;
    puts("Enter name of destination file:");
    s_gets(file_app, SLEN);
    if ((fa = fopen(file_app, "a+")) == NULL) {
        fprintf(stderr, "Can't open %s\n", file_app);
        exit(EXIT_FAILURE);
    }
    if (setvbuf(fa, NULL, _IOFBF, BUFSIZE) != 0){
        fputs("Can't create output buffer\n", stderr);
        exit(EXIT_FAILURE);
    }
    puts("Enter name of first source file (empty line to quit):");
    while (s_gets(file_src, SLEN) && file_src[0] != '\0'){
        if (strcmp(file_src, file_app) == 0)
            fputs("Can't append file to itself\n",stderr);
        else if ((fs = fopen(file_src, "r")) == NULL)
            fprintf(stderr, "Can't open %s\n", file_src);
        else
        {
            if (setvbuf(fs, NULL, _IOFBF, BUFSIZE) != 0){
                fputs("Can't create input buffer\n",stderr);
                continue;
            }
            append(fs, fa);
            if (ferror(fs) != 0)
                fprintf(stderr,"Error in reading file %s.\n", file_src);
            if (ferror(fa) != 0)
                fprintf(stderr,"Error in writing file %s.\n", file_app);
            fclose(fs);
            files++;
            printf("File %s appended.\n", file_src);
            puts("Next file (empty line to quit):");
        }
    }
    printf("Done appending. %d files appended.\n", files);
    rewind(fa);
    printf("%s contents:\n", file_app);
    while ((ch = getc(fa)) != EOF) putchar(ch);
    puts("Done displaying.");
    fclose(fa);
    return 0;
}
void append(FILE *source, FILE *dest){
    size_t bytes;
    static char temp[BUFSIZE]; // allocate once
    while ((bytes = fread(temp,sizeof(char),BUFSIZE,source)) > 0)
        fwrite(temp, sizeof (char), bytes, dest);
}
char * s_gets(char * st, int n){
    char * ret_val;
    char * find;
    ret_val = fgets(st, n, stdin);
    if (ret_val){
        find = strchr(st, '\n');   // look for newline
        if (find)                  // if the address is not NULL,
            *find = '\0';          // place a null character there
        else
            while (getchar() != '\n') continue;
    }
    return ret_val;
}
```

说明

- 程序清单13.5　append.c
- 该程序把一系列文件中的内容附加在另一个文件的末尾
  - 问题：如何给文件传递信息。
  - 可以通过交互或使用命令行参数来完成，先采用交互式的方法
    - 询问目标文件的名称并打开它。
    - 使用一个循环询问源文件。
    - 以读模式依次打开每个源文件，并将其添加到目标文件的末尾
- 细化setvbuf()函数打开目标文件的步骤
  - 以附加模式打开目标文件；
  - 如果打开失败，则退出程序；
  - 为该文件创建一个4096字节的缓冲区；
  - 如果创建失败，则退出程序
- 细化拷贝部分
  - 如果该文件与目标文件相同，则跳至下一个文件；
  - 如果以读模式无法打开文件，则跳至下一个文件；
  - 把文件内容添加至目标文件末尾。

7.9 用二进制I/O进行随机访问

程序清单13.6 randbin.c

```
1.  #define ARSIZE 1000
2.  int main(){
3.      double numbers[ARSIZE];
4.      double value;
5.      const char * file = "numbers.dat";
6.      FILE *iofile;
7.      // create a set of double values
8.      for(int i = 0; i < ARSIZE; i++)
9.          numbers[i] = 100.0 * i + 1.0 / (i + 1);
10.     // attempt to open file
11.     if ((iofile = fopen(file, "wb")) == NULL) {
12.         fprintf(stderr, “%s not open.\n", file);
13.         exit(EXIT_FAILURE);
14.     }
15.     // write array in binary format to file
16.     fwrite(numbers, sizeof (double), ARSIZE, iofile);
17.     fclose(iofile);
18.     if ((iofile = fopen(file, "rb")) == NULL) {
19.         fprintf(stderr, "Could … %s ...\n", file);
20.         exit(EXIT_FAILURE);
21.     }
22.     // read selected items from file
23.     printf("index in the range 0-%d.\n", ARSIZE - 1);
24.     while (scanf("%d", &i) == 1 && i >= 0 && i <
    ARSIZE){
25.         long pos = (long) i * sizeof(double);
26.         fseek(iofile, pos, SEEK_SET);    // go there
27.         fread(&value, sizeof (double), 1, iofile);
28.         printf("The value there is %f.\n", value);
29.         printf("Next index (out to quit):\n");
30.     }
31.     // finish up
32.     fclose(iofile);
33.     puts("Bye!");
34.     return 0;
35. }
```

关键概念

关键概念

- C程序把输入看作是字节流，输入流来源于文件、输入设备（如键盘），或者甚至是另一个程序的输出。类似地，C程序把输出也看作是字节流，输出流的目的地可以是文件、视频显示等
- C如何解释输入流或输出流取决于所使用的输入/输出函数。程序可以不做任何改动地读取和存储字节，或者把字节依次解释成字符，随后可以把这些字符解释成普通文本以用文本表示数字
- 要访问文件，必须创建文件指针（类型是FILE *）并把指针与特定文件名相关联。随后的代码就可以使用这个指针（而不是文件名）来处理该文件
- 要重点理解C如何处理文件结尾。通常，用于读取文件的程序使用一个循环读取输入，直至到达文件结尾。C输入函数在读过文件结尾后才会检测到文件结尾，这意味着应该在尝试读取之后立即判断是否是文件结尾