第八章 数据文件处理技术

刘 卉

huiliu@fudan.edu.cn

主要内容

- C文件概述
- 文件类型和文件类型指针变量
- 常用的文件库函数
- 文件处理程序结构
- 文件处理程序实例

8.1 C文件概述

- 数据的存储和管理
  - 存储信息(数据)时, 将一组相关信息组织成文件.
  - 操作系统将文件存储在计算机的外部存储介质中: 磁带, 磁盘等.
  - 为了管理众多文件, 操作系统维持一个层次状的目录结构, 每个文件都有一个名字, 并被登录在某个目录下 ⇨ 路径 + 文件名.

e.g. C:\Users\comet\Documents\C\2021秋季\Chapter8.pptx

C文件概述

- 操作系统以数据流形式组织数据文件.
- 输入/输出系统：在设备与内存之间传输数据.
  - 输入：设备⇨内存
  - 输出：内存⇨设备
- 为了统一管理数据的输入/输出，有二类特殊文件：

  1．从键盘输入的数据流——标准输入文件

  2．向显示屏/打印机输出的数据流——标准输出文件

二种不同形式的数据流

1）正文文件

- 存储字符的编码—ASCII码的二进制表示.

字符1 | 00110001 | '1'的ASCII码: 0X31, '0'的ASCII码: 0X30

字符串10000000	00110001	00110000	…	…	…	…	…	00110000

7个字符0

2）二进制文件

- 数据按其在内存中的存储形式保存到外存中.

整数1	00000000	00000000	00000000	00000001	0X00 00 00 01
整数10000000	00000000	10011000	10010110	10000000	0X00 98 96 80

正文文件

C:\Users\comet\Desktop\tmp1222.txt - Notepad++

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?

2016.12Python课程教学大纲.doc | tmp1222.txt

Address	0	1	2	3	4	5	6	7	Dump
00000000	31	0d	0a	31	30	30	30	30	1..10000
00000008	30	30	30						000

二进制文件

C:\Users\comet\Documents\C\Examples\Chap8\test.dat - Notepad++

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?

tmp1222.txt | test.dat

Address	0	1	2	3	4	5	6	7	Dump
00000000	01	00	00	00	80	96	98	00	....€?.□

正文文件 *vs* 二进制文件

正文文件的优缺点

- 优点
  - 程序可对文件逐个字符进行处理.
  - 文件中的数据可供人阅读.
- 缺点
  - 数据在内/外存中的存储形式不一致.
  - 在内存中，按数据类型分配存储单元.

    e.g. 整数1和整数10000000均占4bytes.
  - 在外存中，所有数据均以字符形式存储 ⇨ 同类型数值数据所占存储空间大小不一致.

    e.g. 字符1占1byte，字符串10000000占8bytes.

文件输入/输出时，必须进行内外表示形式之间的转换.

二进制文件的优缺点

- 数据按其在内存中的存储形式保存到外存中⇨各类数值数据在文件中的存储字节数固定.

  e.g. 整数1和10000000均占4bytes.

- 文件中的数据人类无法直接阅读.
- 节省外存空间，无需进行数据内外表示形式之间的转换.
- 用于程序与程序/设备之间传递成批数据信息.

程序与文件交换数据的实现过程

- 输入操作：程序从文件中读取数据.
- 输出操作：程序向文件写入数据.
- 对存储设备的读写操作⇨在操作系统的管理和控制下进行.
- 为了能高效地管理和控制设备，操作系统给程序正在使用的每个文件分配两个内存块.

1. 输入/输出缓冲区

- 用于平滑CPU速度和输入/输出速度的差异，减少启动设备的次数.
- 从文件读入信息时，系统每次读入足够多的信息存于缓冲区，供程序逐步取用.
- 写文件时，先把信息暂时写到缓冲区，待缓冲区写满或写文件结束时，才把缓冲区的内容写入文件.

2. 输入/输出控制块

- 存放对文件进行操作所需的控制信息：
  - 文件名
  - 文件读写状态
  - 文件缓冲区的大小和位置
  - 当前读写位置
  - ……

C文件概述

- C语言的文件设施
  - 本身不提供有关文件操作的输入/输出语句.
  - 库函数：实现文件的打开，关闭和读写操作.
- 程序读取文件中的数据
  1. 打开文件：系统为文件分配两个内存块，填写必要的控制信息，并将控制块的指针返回给程序.
  2. 以该指针为实参，调用系统提供的文件库函数，读取数据.
  3. 关闭文件：系统回收两个内存块.

C文件概述

- 程序向文件中写入数据
  - 打开文件(建立一个新文件)：程序得到系统返回的控制块指针.
  - 以该指针为实参，调用系统提供的文件库函数，向文件输出数据.
  - 关闭文件：系统回收两个内存块.

8.2 文件类型和文件类型指针变量

文件输入/输出控制块的信息存储在一个结构变量中, 其类型为FILE.

FILE的定义在stdio.h文件中.

系统内部定义了一个FILE类型的结构数组;

结构数组的大小确定了C程序能同时打开的文件数目.

stdio.h中FILE的类型定义

```
struct _iobuf {
    char *_ptr;
    int _cnt;
    char *_base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char *_tmpfname;
};
typedef struct _iobuf FILE;
```

输入/输出控制块：对文件进行操作所需的控制信息.

文件类型和文件类型指针变量

- FILE指针变量
  - 文件打开函数fopen()为文件指定一个FILE结构数组的元素，并返回该元素的指针.
  - 通过该指针可引用结构中的文件控制信息，正确完成文件操作，并将新的控制信息回填到文件控制块中.
  - 为了使用该文件指针，必须定义一个FILE类型的指针变量保存它.

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

文件类型和文件类型指针变量

- C系统中的标准文件指针
  - 与标准输入/输出设备的数据流对应.

1) stdin: 键盘输入数据流⇨标准输入文件;

2) stdout: 输出到显示器的数据流⇨标准输出文件;

3) stdprn: 输出到打印机的数据流⇨标准打印输出文件;

4) stderr: 标准错误输出文件

8.3 常用的文件库函数

打开/关闭文件函数
fopen & fclose

输入/输出字符函数
fgetc & fputc

格式输入/输出函数
fscanf & fprintf

输入/输出字符串函数
fgets & fputs

打开文件函数

文件控制信息结构指针

文件名(包括目录路径)

文件的读写方式

```
FILE *fopen(char *Filename, char *Mode);
FILE *fp = fopen("\\usr\\smp.dat", "r");
```

'\'需转义

- 以只读方式打开根目录下usr子目录中的smp.dat文件;
- 把存储该文件的控制信息结构的内存单元首地址保存到指针fp.

- 调用fopen()时，可能会出现文件不能打开的情况．原因：
  1）在读方式下打开不存在的文件．
  2）在写方式下，外存已无剩余的自由空间．
  3）外设故障．
  4）超过系统能同时打开的文件数．
  - 文件不能打开时，fopen()返回NULL．
- 考虑到上述极端情况，常用以下形式C代码打开文件：

```
if ((fp = fopen(fname, "r")) == NULL) {
    printf("Can not open %s file.\n", fname);
    return 1;   /* 程序非正常结束 */
}
```

调用fopen()后，立即检查打开是否成功．如果不成功，输出提示信息，结束当前程序段运行．

FILE *fopen(char *Filename, char *Mode)

文件读写方式	意　义
r(b)	只读，为读打开正文(二进制)文件
w(b)	只写，为写建立并打开正文(二进制)文件
a(b)	追加，从正文(二进制)文件末尾开始写
r(b)+	读写，为读/写打开正文(二进制)文件
w(b)+	----，------建立并打开正文(二进制)文件
a(b)+	----，------打开正文(二进制)文件

打开文件函数

- 二进制文件的读写方式：在对应的正文文件读写方式后接上字符'b'，如："rb".
- 用"r"方式打开文件：要求该文件已经存在⇨只能用于从文件获取数据.
- 用"w"方式打开文件：只能用于向文件写入数据.

  1）如果要打开的文件不存在⇨新建一个以指定名字命名的文件;

  2）若文件已存在⇨原文件中的数据全部被清除，准备写入新数据.

打开文件函数

- 用"a"方式打开文件：从原文件的末尾开始添加新数据
  - 向文件写入数据，并且不删除已有数据.
- "r+"，"w+"，"a+"方式：都可以执行输入/输出操作.

  1）"r+"：只允许打开已存在的文件，先读后写；

  2）"w+"：清空/新建一个文件，先写后读；

  3）"a+"：打开一个已存在的文件，从文件末尾添加数据，然后可以读数据.

标准文件的打开

- 标准输入文件stdin，标准输出文件stdout，标准出错输出文件stderr，标准打印输出文件stdprn.
- These stream pointers are constants, and cannot be assigned new values.
- 程序运行时，系统会自动打开这些标准文件，做好输入/输出准备.

关闭文件函数

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

- 使用完一个文件后, 程序应立即关闭它.

  e.g. `fclose(fp);`

  关闭成功, 返回0; 检测到错误, 返回EOF(-1).

1) 终止fp与文件之间的联系: 不能再通过fp对文件进行读写操作, 除非文件被再次打开.

2) 释放为文件所分配的结构和缓冲区——可用来存放新打开的文件.

关闭文件函数

- fp可用来保存新打开文件的文件指针，与新打开的文件相关联.
- 若文件以写方式打开，及时调用fclose()防止数据丢失.
  - 如果程序已完成写文件工作，而数据还未写满缓冲区，程序突然非正常结束，则暂留在缓冲区内的数据丢失.
  - fclose()先把缓冲区中的数据输出到文件，然后才终止fp与文件之间的联系.

fgetc & fputc

- `int fgetc(FILE *_File);`
  - 从与fp相联系的文件中顺序读入下一个字符.

    e.g. ch = fgetc(fp);

  1) 文件未结束, 函数返回读入字符的ASCII码.

  2) 文件结束, 函数返回文件结束标志EOF.
- `int fputc(int _Ch, FILE *_File);`
  - 将ch输出到与文件指针fp相联系的文件中.
  - 输出成功, 返回输出字符的ASCII码; 失败, 返回 EOF.

fgetc & fputc

- 以标准文件指针作参数，调用输入/输出库函数.

```
e.g.  fgetc (stdin);  //getchar()
//从标准输入设备(键盘)读取(输入)1个字符数据
      fputc(c, stdout);//putchar(c)
//向标准输出设备(显示器)输出1个字符数据c
```

文件处理程序的结构

- 文件处理程序的基本步骤

1) 在程序开始处, 定义文件指针变量和存储文件名的字符数组.

```
FILE *fp;
char fname[40];
```

2) 输入文件名.

```
printf("请输入文件名(包括路径, 扩展名)\n");
scanf("%s%*c", fname);   /* 读入文件名及随后的回车符,
                            等价于gets(fname) */
```

文件处理程序的结构

3）使用文件前，按使用要求打开文件.

```
if ((fp = fopen(fname, "r")) == NULL) {
  printf("Can not open %s file.\n", fname);
  return 1;
}
```

4）调用库函数对数据文件进行输入/输出.

5）文件使用结束后，关闭该文件.

正文文件输入的程序结构

```
FILE *fp;
……  /* 说明有关变量并置初值 */
if ((fp = fopen(fname, "r")) == NULL) {
  printf("Can not open %s file.\n", fname);
  return;
}
while ((c = fgetc(fp)) != EOF) {
  …  /*逐个读取文件中的字符进行处理  */
}
fclose(fp);
… /* 输出处理结果 */
```

[例8_1] 统计正文文件中非空白字符个数并输出

```
FILE *fp;
int count;
char ch, fname[40];
puts("输入文件名!");
gets(fname);  //scanf("%s%*c", fname)
if ((fp = fopen(fname, "r")) == NULL){
    printf("Can not open %s file.\n", fname);
    return 1;  /* 程序非正常结束 */
}
count = 0;
while((ch = fgetc(fp)) != EOF) {
    if(ch != ' ' && ch != '\t' && ch != '\n')
        count++;
}
fclose(fp);
printf("文件%s有%d个非空白字母.\n", fname, count);
```

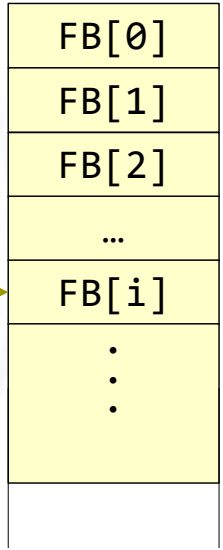
例8-1

- fgetc函数为何能顺序读取文件中的字符?
  - 文件的当前读写位置：在文件的控制信息结构FILE中，有一个成员保存文件的当前读写位置.
  - 每读写一个字符后，若文件还未结束，该成员就自动指向下一个字符⇨反复调用函数fgetc()能顺序读取文件中各个字符.

文件控制块

```
struct _iobuf {
    char *_ptr;
    int _cnt;
    char *_base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char *_tmpfname;
  };
typedef struct _iobuf FILE;
```

文件缓冲区



二进制文件输入的程序结构

```
char c;  /* 或int c */
FILE *fp;
…   /* 说明有关变量并置初值 */
if ((fp = fopen(fname, "rb")) == NULL) {
   printf("Can not open %s file.\n", fname);
   return;
}
while (!feof(fp)) {
   c = fgetc(fp);
   …   /* 对字节数据c进行处理 */
}
fclose(fp);
… /* 输出处理结果 */
```

二进制文件输入的程序结构

- `int feof(FILE *_File);`
  - 判断正文/二进制文件是否结束.
  - 如果结束，返回非零值；否则返回零值.
  - 对于二进制文件，读入的某个字节信息转换成整数后可能是-1
    ⇨ 不能以读入字节的值是否为EOF来判定文件是否结束.

生成文件的程序结构

```
int c;
FILE *fp;
…  /* 定义有关变量并设置初值 */
fp = fopen(fname, "w"); //fp=fopen(fname,"wb")
while (还有字符 / 字节) {
  …  /* 将字符(或字节)存于变量c */
  fputc(c, fp);//将生成的字符(或字节)输出至文件
}
fclose(fp);
…  /* 输出程序结束报告 */
```

文件处理程序实例

[例] 把从键盘读入的字符逐行复制到指定文件中, 直至输入空行结束.
要求:

1) 复制时, 在数字字符序列与其它字符序列之间插入1个空格符;

2) 统计输入字符和输出字符的个数, 并将统计结果输出.

- 为了判定前一个字符和当前字符是否为数字符，引入标志变量pre_d和cur_d. 是数字符时为1, 否则为0.

例8-2

格式输入/输出函数fscanf和fprintf

fprintf(FILE *wfp, 格式字符串, 输出项表)

fscanf(FILE *rfp, 格式字符串, 输入项地址表)

- 作用与函数scanf()和printf()相仿.

- 不同之处:
  - scanf()从stdin中输入数据; printf()向stdout输出数据;
  - fscanf()和fprintf()的输入/输出对象是一般文件⇨多了一个文件指针形参.

格式输入/输出函数

```
    int i = 1;
    float r = 2.0;
    fprintf(wp, "i = %d r = %6.4f\n", i, r);
/*将变量i和r的值按格式输出到与wp关联的文件中*/
    fscanf(rp, "i = %d r = %f", &i, &r);
/*从与rp关联的文件中按输出格式为i和r输入值 */
```

[例8_3] 从键盘输入整数序列，将它们依次保存到表格文件中.

```
int x, k;
FILE *fp;
char fname[40];
printf("请输入表格文件名(以.csv为文件名后缀): ");
gets(fname);
if ((fp = fopen(fname, "w")) == NULL) {  //打开文件为可写
   printf("不能打开文件%s!\n", fname);
   return 1;
}
printf("请输入整数:\n");
k = 1;
while (scanf("%d", &x)) {
   fprintf(fp, "%d,", x);
   if (k++ % 5 == 0)  //文件的每一行保存5个整数
      fprintf(fp, "\n");
}
fclose(fp);
```

例8-3

fgets & fputs

- 正文文件输入/输出字符串

```
char* fgets(char* _Buffer, int _MaxCount, FILE* _Stream);

int fputs(char* _Buffer, FILE* _Stream);
```

  - fgets()从_Stream所指文件中读取字符序列，存于_Buffer所指的存储区域. 连续读入_MaxCount-1个字符或遇到换行符时，读取过程结束.
  - fputs()把_Buffer所指字符串复制到_Stream所指文件. 若发生输出错误, 返回EOF; 否则返回一个非负值.

8.5 文件处理程序实例

[例] 输入一篇英语短文, 统计文件中的行数、单词数和字符数.

- 程序循环读入每个字符, 字符数依次递增, 遇换行符则行数增1.
- 单词: 不能简单地以遇到空白符作为判断单词数增1的条件, 因为可能会连续出现空白符.

  引入一个状态变量state, 在单词中时置1, 在单词外时置0.

例8-5

文件处理程序实例

[例] 逐行复制已知文件，生成新文件．要求新文件是行定长文件．即，每行字符个数一样多．

- 参照已知文件，生成新文件．
  - 同时打开二个文件：1个用于读，1个用于写．
- 程序中进行文件复制时，
  - 若源文件的一行字符数不足定长，则用空白符填充；
  - 若超过定长，则按定长截断为多行进行复制．

例8-6