北京科海培训中心

C 语言实践(二)

一个编辑软件的设计与编制

沈建威　严洪华　倪旭东　编著

清华大学出版社

前　言

文字编辑软件是计算机上使用最频繁的软件之一，一个好的编辑软件可以大大地提高使用者的工作效率。编辑程序往往又是许多优秀软件的组成部分，如人们常用的 dBASE Ⅲ，Turbo C 等都带有自己的编辑器。怎样设计、编制一个编辑软件，是不少软件工作者感兴趣的问题。本书通过讲述怎样用 C 语言编写一个全屏幕中西文编辑软件，使读者既能了解编写字处理软件的思路和方法，又能从中学到许多 C 语言编程的技巧。

用 C 语言编程，比用其它高级语言更能充分地发挥计算机硬件的功能，它又比汇编语言易学、可读性强。C 语言的这些突出优点，吸引了越来越多的人学习和应用 C 语言。随着学习的深入，许多人已经不满足于从一般的 C 语言教科书中了解一些 C 语言的基础知识和编制一些简单的小程序。因此，编写本书的又一目的就是通过对 C 语言编程技巧深入浅出的讲解，帮助读者实现从一个 C 语言初学者到高级程序员的过渡。

本书共分 11 章，第 1 章讲述全屏幕编辑软件的主体结构和设计构思；第 2 章分析汉字的输入及显示的特点和实现方法；第 3 章介绍全屏幕编辑软件的屏幕显示和主要显示模块的设计编制；第 4 章介绍文件目录列表、文件名输入和辅助文件名的建立；第 5 至 11 章讲述各种编辑功能函数的编制。

本书最后提供了一个用 Turbo C 编写的完整的全屏幕字处理软件 BJ 的源程序清单，我们几乎在每一条语句后面都加了中文注释，以帮助读者比较清晰地了解各语句的含义及功用。当你完全读懂了这个程序后，一定会感到受益匪浅。如果把这个程序输入到微机中，并且用 Turbo C 2.0 编译、连接、自己认真调试成可执行文件，得到的就是一个非常实用的全屏幕文字编辑软件。这个软件能实现文字的输入和删除，文本行的插入和删除，上下翻屏，左右移屏，字符串的查找和替代，文本排版，字块的设置、删除、移动、拷贝、写盘，外部文件的加入，文本的打印等功能，并可用键盘右侧的数字小键盘输入中文制表符。这样，通过一次完整的编程实践，你对 C 语言，对汉字软件的编程技术，对 DOS 的编程环境，对编辑软件的一般特性，以至于对软件工程的基本方法，都会有一个更切身的体验。

由于这不是一本讲述 C 语言基础知识的入门教材，所以我们没有从 C 语言最基本的概念说起。对于刚刚接触 C 语言的读者，建议能找一本 Turbo C 的入门书作为参考，对照着阅读本书，这样对迅速掌握 C 语言是很有好处的。如果你已是一个熟练的 C 语言程序员，这本书也一定会给你许多新的启发。本书中的许多设计思想和子函数模块，在用 C 语言编制其他软件时也是非常有用的。

本书既可以供广大计算机软件设计人员在设计软件时参考，又可作为高等院校计算机软件专业的教学参考书，对于广大电脑爱好者，更是一本不可多得的好书。由于水平所限，书中难免有错误和不当之处，恳请广大读者批评指正。

作　者

1994 年 4 月

目　录

第 1 章　设计全屏幕文字编辑软件的总体构思

文字编辑软件是使用最为广泛的一种计算机软件，它经历了一个不断发展的过程，已经日趋完善。目前使用最为普遍的文字编辑软件有两类，一类是行编辑软件，如大家熟知的 EDLIN 就属此类。行编辑程序的编辑以行为单位进行，对某一行的操作通过输入各种编辑命令来执行。对当前行中字符的编辑，操作者通常通过一些功能键配合字符输入来完成。另一类编辑软件是全屏幕编辑软件，操作者通过编辑命令、功能键或屏幕菜单选择执行各种编辑操作。全屏幕编辑软件比行编辑软件更方便灵活，功能更丰富。

C 语言是一种结构化、模块化的程序设计语言，比其他高级语言更贴近计算机硬件。C 语言是又一种编译性的语言，具有生成的目标代码紧凑、效率高、程序运行速度快的特点。它有着丰富的数据类型、运算功能，能直接对位、字节、字和指针进行操作。在其功能强大的函数库中，包含了各种各样的输入／输出 (I／O)、字符串处理、文件管理、数据转换、内存管理等效率很高的函数，用 C 语言来编制中西文编辑软件无疑是非常合适的。

本书通过对一个中西文全屏幕编辑软件 BJ 从总体设计到用 C 语言编制各功能模块的详细讲述，帮助读者了解全屏幕编辑软件的设计编制方法，并领会 C 语言的编程技巧。

1.1　全屏幕编辑软件的基本框架

计算机软件是信息处理的工具，一个软件大体上应包括信息的输入、加工处理和输出三个方面，编辑软件也不例外。

编辑软件工作的主要对象是文本文件，编辑操作可能是对一个已有的老文件的增删修改，也可能是建立一个新文件，因此输入文件名是编辑程序首先要进行的工作。

文本文件是许多个由字符组成的字符串行构成的，字符串行中每个字符的位置称为列，要对文本文件进行各种编辑操作，就必须在内存中开辟一个编辑缓冲区，如果编辑的是一个老文件，还必须把这个文本文件或其中的一部分读入这个缓冲区。

全屏幕编辑是通过屏幕这个“窗口”观察文本和进行编辑操作的，还要利用屏幕提供有关信息和进行人机对话，所以编辑软件的设计，离不开对编辑屏幕显示的设计布局。程序进入编辑状态时，要先显示一个初始屏幕。

在编辑过程中，要选择和实施各种功能的编辑操作，就必须建立各种编辑操作的功能模块，并提供选择它们的手段。在编辑工作终了时，要保存编辑工作的成果，并通过适当的途径退出运行。

以上几项构成了一个全屏幕编辑软件最基本的部分，可以用一个框图来描述全屏幕编辑软件的基本框架(见图 1.1)。

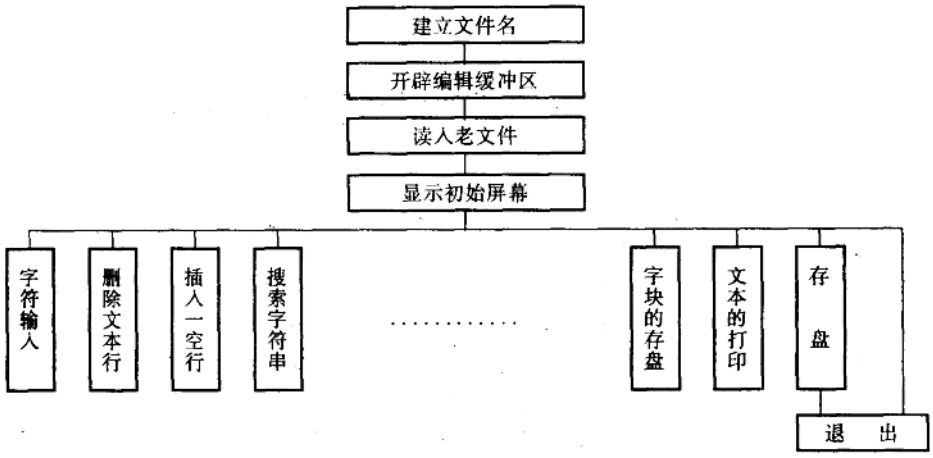


图 1.1　全屏幕编辑软件的基本框架

在实际工作中常常会出现这样的情况：一方面，用户对软件功能越来越高的要求，致使一些软件越来越复杂、庞大；另一方面，用户又希望操作尽可能的简单，这又要求软件不能太复杂。怎样处理好这一对矛盾，根据实际需要找出最佳的方案是软件设计者首先要认真考虑的问题。总之，编辑软件设计时设置功能的多少应视用户的需要而定，在操作方法上要力求简单、高效。

1.2　主要全局变量简介

本书所示例的编辑程序 BJ 中使用的全局变量较多，这是根据该编辑程序的特点所确定的。这些全局变量定义在程序前部，它们的作用域是整个程序。使用较多的全局变量避免了调用函数时过多的参数传递，编程也相对容易些。又因为全局变量在程序中各函数里的表达形式相同，所以在某种程度上增强了程序的可读性。其缺点是削弱了一些与全局变量有关的函数的独立性，因此用 C 语言编程通常不主张用过多的全局变量。

为了便于读者阅读本书中的程序，下面把 BJ 全屏幕编辑程序使用的主要全局变量作一简要介绍：

ss[ ][ ]	编辑缓冲区数组（简称编辑数组）。
xx	文本行号，是当前行在文本中的绝对行号。
yy	文本列号，既是当前列在文本中的列号，又是编辑数组的列下标。
ss_x	数组行号，当前行在编辑数组中的行下标。
ss_max	编辑数组实际存有文本行的最大行号。
x	显示行号，当前行在屏幕上显示的屏幕行坐标。
y	显示列号，当前列在屏幕上显示的屏幕列坐标。
m	左右移屏屏号，首屏为 0。
enq	排版行宽度。
ksx	字块块首的文本行号，未定义块时置为 -1。

ksy	字块块首的文本列号，未定义块时置为 −1。
kwx	字块块尾的文本行号，未定义块时置为 −1。
kwy	字块块尾的文本列号，未定义块时置为 −1。
oa	临时文件1读写起始位置数组 wra[ ] 下标。
ob	临时文件2读写起始位置数组 wrb[ ] 下标。
wra[ ]	临时文件1读写地址数组。
wrb[ ]	临时文件2读写地址数组。
fp_rd	老文件读出的最后行在文本中的绝对行号。
ttl_x	文末行行号，文本最后一行在文本中的绝对行号，先设一个足够大的数作初值，待老文件全部读出后用已读出行数 fp_rd 代替。
ser	当前光标处的字序数。
old	文件标志，old＝0 为新文件，old＝1 为老文件。
fp_end	老文件读完标志，fp_end＝0　未读完，fp_end＝1　已读完。
ins	编辑状态标志，ins＝0　非插入状态，ins＝1　插入状态。
vid	显示方式标志，vid＝0　字符方式，vid＝1　图形方式。
tab	中文表格线标志，tab＝0　非表格线状态，tab＝1　表格线状态。
blck	字块定义标志，blck＝0　未定义块，blck＝1　仅定义块首，blck＝2　已定义块。
chg	文件修改标志，chg＝0　文件未修改，chg＝1　文件已修改。
* mem	长城字符方式显存基本区地址指针。
* mmm	长城字符方式显存扩展区地址指针。
* ddd	暂时存放字块内容的缓冲区。
* hsz	提问行输入字符串的指针。
* fnd	存放要搜索字符串的指针。
* mfile	存放正文文件名的字符串变量。
* bfile	存放后备文件名的字符串变量。
* file1	存放临时文件 1（带扩展名 1）文件名的字符串变量。
* file2	存放临时文件 2（带扩展名 2）文件名的字符串变量。
* fp	正文文件 mfile 的文件指针。
* fp1	临时文件 1 的文件指针。
* fp2	临时文件 2 的文件指针。
* fp3	外部文件或字块写盘文件指针。

1.3　编辑程序的主控模块

根据第 1.1 节提供的全屏幕编辑程序基本框架，可以编制编辑程序的主控函数 main( )。main( ) 先建立文件名和临时文件名、后备文件名，为编辑文件开辟一个编辑缓冲区，如是老文件就读入一定数量的文本行至编辑数组，然后显示编辑屏幕。编辑操作的选择是通过判断击键的键值，确定是输入字符还是执行某种特定的操作，然后用 switch( ) 开关语句，分别作出相应的处理。除了要对文本进行增、删、修改等改变外，还要在屏幕上进行相应的显示。整个操作过程是循环进行的，每一种操作都建有相应的子模块。操作完成后，按 F1 存盘退出，并删除临时文件。因为各种中文操作系统较多地使用了 Ctrl 键或 Alt 键和 F1～F10 中一些功能键的组合，所以，BJ 程序里采用 Shift 键与 F1、F4 等功能键组成复合键，以尽可能地不和各类汉字操作系统发生冲突。

```
#define HH 24                    /* 屏幕显示总行数 */
```

```
#define H1 (HH-1)                          /*信息行屏幕行号*/
#define QB 500                             /*编辑数组最大行数*/
#define HC 255                             /*编辑数组每行最大字节数控制值*/
union inkey {                              /*定义一个存放击键值的联合*/
    unsigned char ch[2];
    int ii;
} cc;
.....
main(int argc,char *argv[])                /*主控函数*/
{
    int i;
    clss(0,HH);                            /*清全部屏幕屏*/
    mod( );                                /*测显示模式*/
    for(i=0;i<QB;i++) {                    /*循环QB次*/
        ss[i]=malloc(HC);                  /*给数组行分配内存空间*/
        *ss[i]=0;                          /*清为空串*/
    }
    mfile=malloc(16);                      /*为正文文件名字符串变量分配内存空间*/
    *mfile=0;                              /*字符串清为空串*/
    if(argc>1) mfile=argv[1];              /*如进入BJ时带形参argv[1],赋给mfile*/
    filename( );                           /*输入文件名，建立临时文件名，老文件读入*/
    hsz=malloc(HC);                        /*为输入提问的字符串变量hsz分配空间*/
    fnd=malloc(HC);                        /*为搜索字符串分配内存*/
    disp_t( );                             /*显示编辑屏幕*/
    Ins( );                                /*定初始状态为"插入"状态*/
    while(1) {                             /*编辑操作的主循环*/
        xh( );                             /*显示当前行、列、序号*/
        coord( );                          /*显示标尺行*/
        goto_xy(x,y);                      /*定屏幕光标位置*/
        cc.ii=bioskey(0);                  /*将按键扫描码读入一联合中*/
        if(cc.ch[0]) {                     /*如果击键扫描码低位不为0*/
          switch(cc.ch[0]) {               /*判断低位字节*/
            case 13:  Enter( );            /*输入回车键*/
                      break;               /*跳出开关语句*/
            case 8:   Del( );              /*退格键删字*/
                      break;               /*跳出开关语句*/
            case 14:  Ctrl_N( );           /*Ctrl+N插入行*/
                      break;               /*跳出开关语句*/
            case 25:  Ctrl_Y( );           /*Ctrl+Y删除行*/
                      break;               /*跳出开关语句*/
            case 20:  Ctrl_T(1);           /*Ctrl+T删至行尾*/
                      break;               /*跳出开关语句*/
            case 5:   Ctrl_E(1);           /*Ctrl+E删至行首*/
                      break;               /*跳出开关语句*/
            case 6:   Ctrl_F(1);           /*Ctrl+F移至块首*/
                      break;               /*跳出开关语句*/
            case 22:  Ctrl_V( );           /*Ctrl+V移动块*/
```

```
                break;                  /*跳出开关语句*/
        case 11:  Ctrl_K( );            /*Ctrl+K 块拷贝*/
                break;                  /*跳出开关语句*/
        case 23:  Ctrl_W( );            /*Ctrl+W 块存盘*/
                break;                  /*跳出开关语句*/
        case 4:   Ctrl_D( );            /*Ctrl+D 删除块*/
                break;                  /*跳出开关语句*/
        case 16:  Ctrl_P( );            /*Ctrl+P 打印文本文件*/
                break;                  /*跳出开关语句*/
        case 18:  Ctrl_R( );            /*Ctrl+R 外部文件插入*/
                break;                  /*跳出开关语句*/
        case 27:  Esc( );               /*ESC 键*/
                break;                  /*跳出开关语句*/
        case 3:   bk( );                /*Ctrl+C 退回DOS系统*/
        default:  Chr( );               /*如为字符键，输入字符*/
    }
  }
  else {
    switch(cc.ch[1]) {                  /*如击键为特殊键*/
        case 81:  PgDn( );              /*PgDn 键，向下翻屏*/
                break;                  /*跳出开关语句*/
        case 73:  PgUp( );              /*PgUp 键，向上翻屏*/
                break;                  /*跳出开关语句*/
        case 59:  F1(1);                /*F1 键，存盘退出*/
                break;                  /*跳出开关语句*/
        case 84:  Shift_F1( );          /*Shift+F1 键，存盘不退出*/
                break;                  /*跳出开关语句*/
        case 60:  F2( );                /*F2 键，移到指定行*/
                break;                  /*跳出开关语句*/
        case 61:  F3( );                /*F3 键，输入格式文件排版行宽*/
                break;                  /*跳出开关语句*/
        case 62:  F4( );                /*F4 键，从光标处至本段末排版*/
                break;                  /*跳出开关语句*/
        case 87:  Shift_F4( );          /*Shift+F4 从当前光标处至文末连续排版*/
                break;                  /*跳出开关语句*/
        case 63:  F5( );                /*F5 搜索字符串*/
                break;                  /*跳出开关语句*/
        case 88:  Shift_F5( );          /*Shift+F5 继续搜索*/
                break;                  /*跳出开关语句*/
        case 64:  F6( );                /*F6 字符串替代*/
                break;                  /*跳出开关语句*/
        case 65:  F7( );                /*F7 定义块首*/
                break;                  /*跳出开关语句*/
        case 66:  F8( );                /*F8 定义块尾*/
                break;                  /*跳出开关语句*/
        case 90:                        /*Shift+F7*/
        case 91:  Shift_F7( );          /*Shift+F8 取消块定义*/
```

```
                break;                  /*跳出开关语句*/
            case 67:  F9( );            /*F9 移到文首*/
                break;                  /*跳出开关语句*/
            case 92:  Shift_F9( );      /*Shift+F9 移到文末*/
                break;                  /*跳出开关语句*/
            case 68:  F10( );           /*F10 表格线功能开关转换*/
                break;                  /*跳出开关语句*/
            case 77:  Right( );         /*→键，右移光标*/
                break;                  /*跳出开关语句*/
            case 75:  Left( );          /*←键，左移光标*/
                break;                  /*跳出开关语句*/
            case 72:  Up( );            /*↑键，上移光标*/
                break;                  /*跳出开关语句*/
            case 80:  Down( );          /*↓键，下移光标*/
                break;                  /*跳出开关语句*/
            case 82:  Ins( );           /*Ins 键，插入/非插入状态转换*/
                break;                  /*跳出开关语句*/
            case 83:  Del( );           /*Del 键，同退格键*/
                break;                  /*跳出开关语句*/
            case 71:  Home( );          /*Home 键光标移到行首*/
                break;                  /*跳出开关语句*/
            case 79:  End( );           /*End 键光标移到行末*/
                break;                  /*跳出开关语句*/
        }
    }
  }
}
```

QB、HC、HH、H1 等常量标识符在程序开头用宏定义命令 #define 定义，程序编译预处理时，凡在程序中出现 QB 的地方都用 500 代替。这样做便于修改，要修改这个常量，只要修改预处理语句一处就可以了。H1 用一个数学表达式 (HH-1) 定义，加括号是基于安全性的考虑。

1.4 在内存中开辟一个编辑缓冲区

编辑缓冲区可以有多种形式，如一维的字符型指针、链表等，在本文中，采取二维数组形式。根据文本文件是由行和列组成的矩阵，我们很容易联想到，可以建立一个二维字符型数组作为文本文件的编辑缓冲区，把要编辑的文本或其一部分放在这个二维数组中。设这个数组为 ss[i][j]，下标 i 对应于数组的行，下标 j 对应于数组的列。i 和 j 都应从 0 开始计数。如 ss[5][7] 即表示第 5 行的第 7 列，而 ss[0][0] 则表示第 0 行的第 0 列。

例如，定义一个 500 行 (0～499)，每行 255 个字节 (0～254) 的二维数组，由于整个数组占用内存很大，是跨段的，因此不能简单地用"char ss[500][255];"来定义，而要用一个指针数组 *ss[500] 来替代，并用库函数 malloc( ) 为其分配内存空间：

```
#define QB 500
```

```
#define HC 255
char * ss[QB];
.....
int i;
for(i = 0;i < QB;i++)    ss[i] = malloc(HC);
```

这样，开辟的编辑缓冲区占用的内存空间即为：

500 × 255 = 127500 (字节)

编辑缓冲区数组的大小往往受到使用环境的限制，如果中文操作系统留给用户的空间较小，可以适当减小 QB 的值。这个指针数组的元素可用 *(ss[i]+j) 表达，也可等价地用数组下标方式 ss[i][j] 来表达，这里采用数组下标表示，是考虑到这样比用指针形式程序清晰性好些，更便于读者阅读。

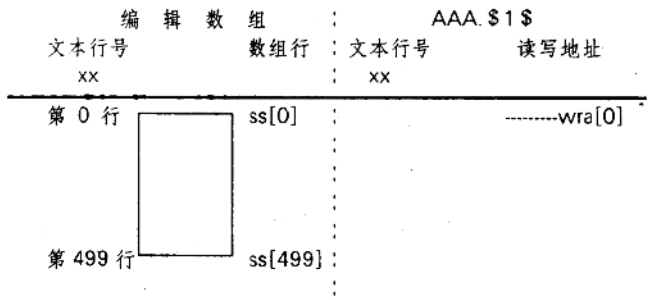
根据以上定义的编辑缓冲区数组，如果编辑的文本文件总行数不超出 500 行，每行又不超过 255 字节(包括软回车符)，则完全可以在这个编辑缓冲区中进行操作。许多较简单的编辑器，在使用时限定被编辑文本文件的大小，就是以在编辑缓冲区能装下为前提的。

1.5 建立临时文件

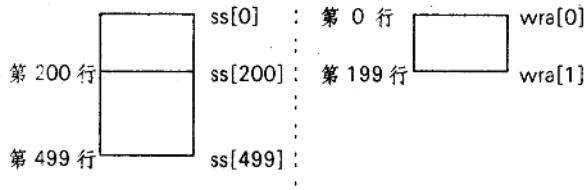
在内存中，操作系统提供的用户空间是有限的，编辑程序自身也要占用一定的空间，因此，编辑缓冲区不可能无限制地设置得很大，如果要编辑一个相当大的文件，不可能全部放在编辑缓冲区的数组内时怎么办呢? 在这种情况下，就只能把文本文件的一部分放在内存中的编辑缓冲区内进行处理。由于在编辑过程中，要进行各种各样的操作，当前操作位置在文本中是经常变动的，因此，编辑缓冲区中的数组 (以下简称编辑数组) 相对于文本文件就必须是动态的。在移动的过程中，必然有一部分已编辑处理过的文本需要临时有个地方放一放，以腾出空间再读入另一部分文本。这时就必须借助于外部存储器——磁盘，在磁盘上建立两个临时文件，一个用于存放编辑数组中的文本内容之前的已读入过编辑数组的文本行，另一个用来存放编辑数组中文本内容后的已读入过内存的文本行。

为了便于叙述，不妨假设我们在编辑一个名为 AAA.TXT 的文本文件，这需在磁盘上建立 AAA.1 和 AAA.2 两个临时文件。假定 AAA.TXT 是一个新文件，并已定义了一个 500 行的字符串数组 ss[i][j] 作为编辑缓冲区。开始编辑时，新建立的 AAA.TXT 是一个空文件，编辑数组也是空的。我们通过键盘不断把文本输入这个编辑数组，从 ss[0] 行起，写到了 ss[499]，这时文本的第 0～499 行就放在数组项 ss[0]～ss[499] 中。如果继续换行输入，编辑数组就放不下了，怎么办呢? 这就要用到临时文件了。我们可将文本第 0 行～第 199 行的共 200 行从编辑数组中取出来，放到临时文件 AAA.1 中去，并把编辑数组中的第 200～499 行依次前移到 ss[0]～ss[299] 中，这样编辑数组的后半部就腾出 200 行空间，可供继续输入了。当又输入 200 行后，再从编辑数组前部放 200 行到 AAA.1 中去，编辑数组中留下的是文本第 400～699 行。在此过程中，要建立一个数组 wra[oa] 来存放 AAA.1 每一次读写的起始位置相对于文件头的偏移量，即 wra[0]～wra[1] 为最早放入 AAA.1 的 200 行(文本的第 0～199 行)，wra[1]～wra[2] 之间为第二

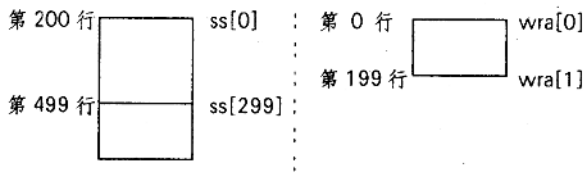
次放入的内容(第 200～399 行)，wra[2]为下一次写入的起始地址。图 1.2 是上述过程的示意图。



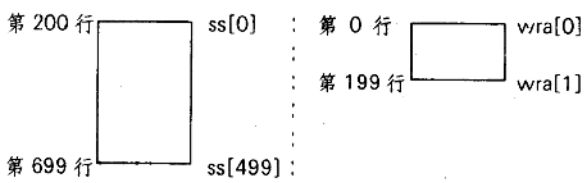
(1) 从键盘输入 500 行至编辑数组，AAA.1 为空文件



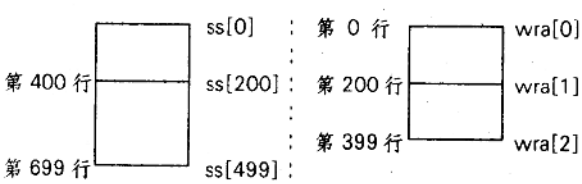
(2) 从编辑数组前部移 200 行至临时文件 AAA.1



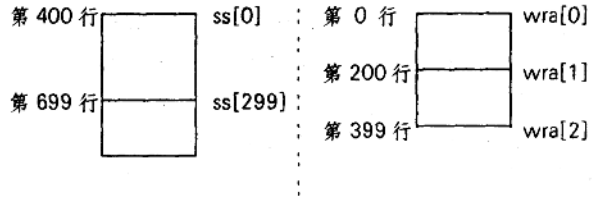
(3) 编辑数组中剩余 300 行前移



(4) 再从键盘输入 200 行至编辑数组



(5) 将编辑数组前 200 行移至 AAA.1



(6) 编辑数组中剩余 300 行前移

图 1.2 新文件的输入过程

从编辑数组移指定行数到临时文件 1 是通过函数 wfp1( )完成的:

```
#define Q3 (QB * 2 / 5)                    /* 每次读写临时文件的行数 */
.....
wfp1( )                                    /* 将内存数组上部 Q3 行写入临时文件 1 */
{
  fseek(fp1,wra[oa],SEEK_SET);             /* 指针移到写入起始位置 */
  write_to(0,Q3,fp1);                      /* 写 Q3 行到 fp1 */
  wra[++oa]=ftell(fp1);                    /* 记录下一次写入起始位置 */
  movbk(0,Q3);                             /* 剩下的数组前移 Q3 行 */
  ss_x-=Q3;                                /* 当前行数组行号减少 j */
  ss_max-=Q3;                              /* 数组行最大行号减少 j */
}
```

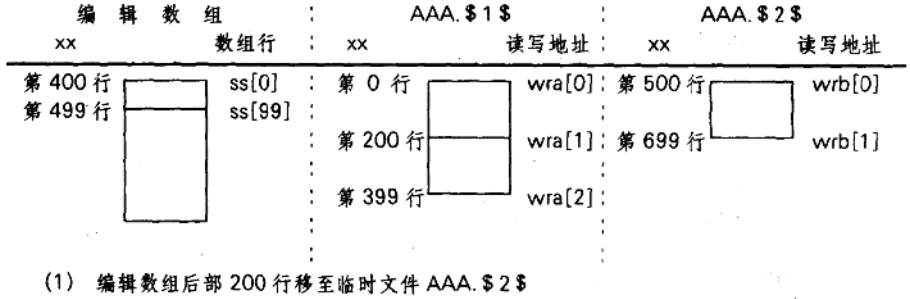
库函数 fseek( )把本次写入 fp1 的起始位置指针移至数组 wra[oa]记录的位置，参数 SEEK_SET 为从文件头开始按给定的偏移量移动文件读写指针位置。用库函数 ftell( )测出临时文件 fp1 写入指定行后的读写指针位置，并记入数组 wra[oa]中，oa 已用增量运算符 ++ 增 1。wfp1( )里调用了函数 write_to( )，它的作用是把编辑数组中从指定行起的一定数量的文本行写入一个文本文件中去。

```
write_to(int a,int b,FILE * f)             /* 把数组第 a 行起的 b 行写入文件 f */
{
  int i;
  .....
  for(i=a;i<a+b;i++) {                     /* 从 a 行起循环 b 次 */
    fputs(ss[i],f);                        /* 将该行字符串写入文件 f */
  }
  .....
}
```

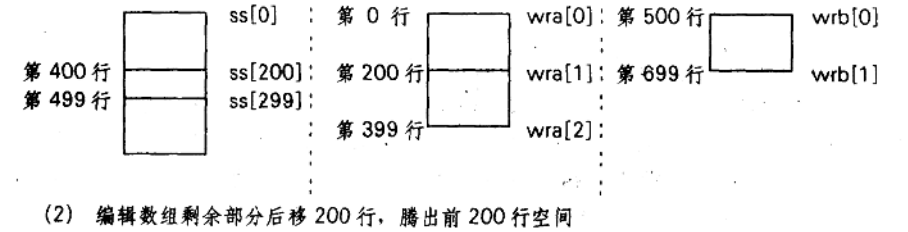
数组中各行前移是由函数 movbk( )实现的:

```
movbk(int s,int a)                         /* 数组 s+a 行起前移 a 行 */
{
  int i;
  for(i=s;i<QB-a;i++) strcpy(ss[i],ss[i+a]);   /* 前移 a 行，覆盖原行 */
  while(i<QB) * ss[i++]=0;                 /* 编辑数组剩余各行清为空行 */
}
```

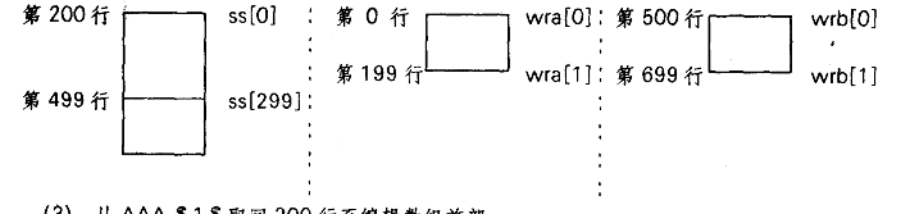
这样似乎我们已经突破了 500 行编辑数组的限制，可以编辑长文件了，但是事实上并不是如此简单。如果这时我们又需要回过头来修改第 100 行的地方，而现在编辑数组里装的是第 400～699 行，又怎么办呢？这时就必须用到第二个临时文件 AAA.2，先把文本第 500 行～第 699 行的共 200 行从编辑数组中取出放入 AAA.2，将编辑数组中剩余的第 400～499 行下移到 ss[200]～ss[299] 处，然后把 AAA.1 中从偏移地址 wra[1] 起的 200 行读出放入编辑数组 ss[0]～ss[199] 处，AAA.1 的当前读写地址由 wra[2] 变为 wra[1]。对 AAA.2 的读写位置也要建立一个数组 wrb[ob] 来存放，它的当前读写位置已由初始的 wrb[0] (等于 0)，变成 wrb[1]。这时编辑数组中放的文本行是第 200～499 行。还要再进行一次这样的过程才能把第 100 行移入编辑数组(如图 1.3)。



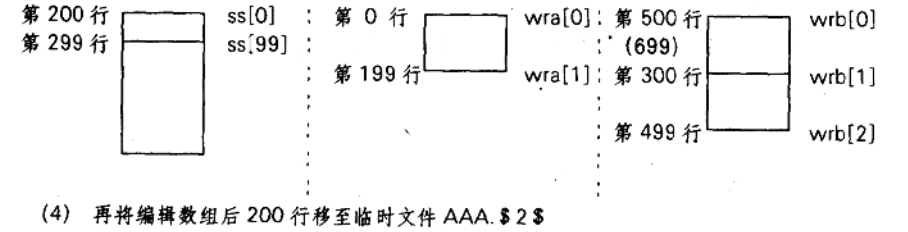
(1) 编辑数组后部 200 行移至临时文件 AAA.2



(2) 编辑数组剩余部分后移 200 行，腾出前 200 行空间



(3) 从 AAA.1 取回 200 行至编辑数组前部



(4) 再将编辑数组后 200 行移至临时文件 AAA.2

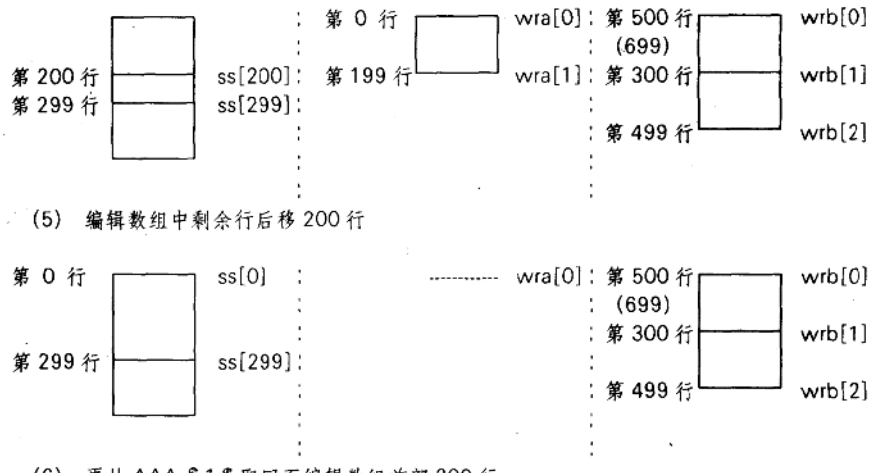


图 1.3 编辑内存数组移动时临时文件的作用

两个临时文件的读写都遵循"先入后出、后入先出"的原则。要实现以上的各过程，就有必要建立从编辑数组中取出指定行数写入文件、从文件中读出一块记录放入编辑数组指定区域和把编辑数组中的行移动位置的子模块。

编辑数组取指定行写入临时文件 2 时调用了函数 wfp2( )，它和 wfp1( ) 的区别在于：wfp1( ) 是从编辑数组前部取 Q3 行放入 fp1，而 wfp2( ) 则是从编辑数组后部取 Q3 行写入 fp2 。

```
wfp2( )                                    /*写 Q3 行到临时文件 2 */
{
  int i;
  fseek(fp2,wrb[ob],SEEK_SET);             /*定 fp2 指针到本次读写位置 */
  write_to(ss_max-Q3+1,Q3,fp2);            /*从数组 ss_max-Q3+1 行起,写 Q3 行到 fp2 */
  if(xx-ss_x+ss_max==ttl_x) fputc(0,fp2);  /*文末行以'\0'定界 */
  wrb[++ob]=ftell(fp2);                    /*记录下一次写起始地址到指针数组 wrb[ ] */
  ss_max-=Q3;                              /*数组行相应减少 */
  for(i=ss_max+1;i<QB;i++) *ss[i]=0;       /*数组后部未用行初始化 */
}
```

从一个文件里读出指定行文本放入编辑数组指定区域的函数为 read_from( )，该函数从文件中当前读写位置逐行读出，放入编辑数组。函数 read_from( )的返回值是实际读出的行数。read_from( )前的类型说明 int 也可省略，因为 C 语言默认的缺省类型为 int。

```
int read_from(int a,int b,FILE *f)         /*从文件f读入b行放入数组ss[]的第a行起 */
{
  int i,j;
  .....
```

```
    for(i=a;i<a+b;i++){                    /*逐行读出*/
      if(fgets(ss[i],HC,f)==NULL){         /*从f读出一行,放入数组,如读失败*/
        j=0;
        while(ss[i-1][j]){                 /*逐字节对比文末行,字符值为真(非0)则循环*/
          if(ss[i-1][j]==0x1A){            /*如为文件结束符*/
            ss[i-1][j]=0;                  /*用'\0'替代*/
            break;                         /*跳出while循环*/
          }
          j++;                             /*后移一字节*/
        }
        break;                             /*跳出for循环*/
      }
    }
    .....
    return i-a;                            /*返回读出行数*/
}
```

程序中的 while(ss[i-1][j]) 是 C 语言风格的写法，相当于 while(ss[i-1][j]!=0)，其意思是 ss[i-1][j] 为"真"则循环。C 语言进行判断时把不等于 0 称为"真"，等于 0 称为"假"。在后面的程序中，还可以看到例如把表达式 if(chg!=0) 写成 if(chg)，把 if(cc.ch[0]==0) 写成 if(!cc.ch[0]) 的例子。

文本行的读出用库函数 fgets( ) 实现，从上述程序中可以看出，fgets( ) 带有三个形参：第一个是放读出字符串的字符型数组，第二个是读出的最大字符数 HC(=255)，第三个是读写的文件指针。fgets( ) 对行的判断是以换行符 0x0A 或空字符 '\0' 为界的，在从当前读写位置起的 HC 个字符中没有 0x0A 或 '\0' 时，它就从 f 中读出 HC 个字符，如读出过程中遇到这两种符号之一，则读出的字符串到此为止，fgets( ) 返回指向串的指针。当文件读完时，如继续读，则 fgets( ) 返回 NULL。NULL 的值为 0，定义在头文件 stdio.h 中。

在 read_from( ) 中对文末行进行了处理，去除了文件结束符 0x1A。从文件读出的过程可以看出，BJ 编辑的文本文件在编辑数组中，除文末行以 '\0' 结尾，其余各行均以硬回车换行符 0x0D0A 或软回车换行符 0x8D0A 并加 '\0' 结束。

1.6 一个老文件的编辑过程

如果是编辑一个已有的老文件 BBB.TXT，这个文件又比较长，通常总是先把文首开始的一部分放到编辑数组中，同时又给编辑数组留一定的空间，如读入 300 行，留 200 行空间。这样做的好处是：在编辑过程中对编辑区数组插入一些行或删除一些行时，只须对编辑数组中的部分行进行移动，而不必对整个文本文件进行重新排列，而且，只须在编辑区域将要移出内存数组、因增删而使编辑数组将发生溢出或行数太少将影响屏幕显示时才对原文件或临时文件进行读写，而不必频繁地读盘、写盘。

假定老文件 BBB.TXT 有 1001 行(行号 0～1000)，两个临时文件分别是 BBB.1 和 BBB.2，开始时从 BBB.TXT 读 300 行放入编辑数组中。这时如要修改第 600 行的内