第11卷 第1期 郑州工学院学报 Vol.11 No.1
1990年 3 月 Journal of ZhengZhou Institute of Technology Feb. 1990

系统行编辑的功能扩充与设计*

王文义 孙立贤

(计算机及自动化系)

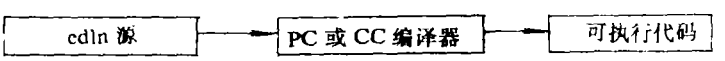
摘 要: 本文对计算机系统软件--行编辑程序的基本构造与实现方法进行了讨论，并对其命令集进行了联想式的简化设计，其效果是命令数减少、功能增强。这个工作对于理解逐步摒弃传统手工编制方法、使用软件工具自动生成更符合用户工作环境或更高一级的系统有关应用程序、逐步实现软件的工程化设计有一定意义。

关键词: 行编辑程序，编辑命令集

行编辑程序是计算机系统软件中最基本的应用程序之一，由于它的命令集合较小，因此具有简捷易学、容易掌握等特点。

系统软件，对普通用户来说，总有一种神秘的感觉。形成这种看法是有其历史原因的。因为用户通常购得的计算机并不总是附有系统中各种软件的源程序（若有，则另外计费）以供用户剖析与学习的，而系统软件的各种可执行码一般又加了密，即便不加密，庞大的机器代码群在短时间内也是难于理解的。因此要想掌握某些系统软件的构造技巧和设计思想，就势必会遇到困难。

现在的计算机一般都配有 PASCAL、C 等自编译语言，可以用它们直接书写各种系统程序，这些编译器为扩充系统软件资源提供了可能性。PASCAL 语言和 C 语言具有小巧连贯、结构性好的特点，因此它们能够描述和解决计算机系统程序设计中所遇到的复杂结构。著名的 UNIX 系统绝大部分就是用 C 语言书写的。鉴于此，我们也可以使用它们设计一个功能较强的、使用较少命令的行编辑程序，并使它成为实用的机器代码。在编制的全过程中，我们可以窥到编辑程序的很多技巧。行编辑的产生过程如下:

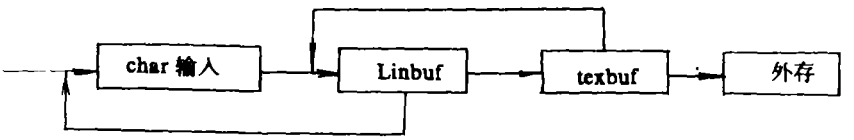


在设计过程中，我们拟采用 PASCAL 语言作为主要的实现工具，当然也涉及到了 C 语言和汇编语言。

* 收稿日期: 1989.03.26

1 对缓冲区的考虑

设计两个缓冲区 (Buffer)：行缓冲区 Linbuf 和正文缓冲区 texbuf。我们把这两个缓冲区以及外部存贮器理解为阶梯形结构，Linbuf 在上，texbuf 居中，外部存贮器在下。执行过程描述为：先往 Linbuf 中输入内容，放满后复制入 texbuf，释放 Linbuf，为下次输入做好准备。一旦 texbuf 放满或用户结束输入，则把 texbuf 中内容全部复制到外部存贮器中。整个过程如下图反示。



1．1 行缓冲区 Linbuf：由于行编辑的操作对象主要以正文行为主，因此关于行的因素要多考虑一些。其中考虑的主要因素就是设置行首、行尾的标志问题。考虑到一般 CRT 显示位置每行多为 80 个字符（当然根据实际情况可适当增减），因此可设置 Linbuf 的最大容量为 80 个单元。这一步比较容易实现，因为是线性的，只在设置一个指示器去跟踪字符在正文行中的当前位置即可，最简单的办法是定义一个一维数组，其下标本身就是一个指示器。

如何设置行首和行尾的标志？不少 PASCAL 版本都设有标准库函数 EOLN、CHR 和 ORD 等，其中 EOLN 是对标准的输入／输出文件 INPUT 和 OUTPUT 而言的，文件的成分只能是那些 ASCII 码中的可显示字符与少数控制字符。对于回车和换行等控制字符，由于无法用某个可见的字符去表示它们，因此只有求助于函数 CHR 和 ORD，即用 CHR(10)表示换行符 LF，用 CHR(13)表示回车符 CR，至于正文输入结束与否，则可使用 ORD 函数值是否等于 4 加以判别。

UNIX 系统中的 SVS PASCSL 使用的是经扩充的 ASCII 字符集，共有 256 个字符，其序号范围为'／＄00'…'／＄off'，它允许标准函数的参数是控制字符，这些字符虽然不可显示，但是作为一个函数结果实体存在某个单元中是可行的。在输出该单元的内容时，其执行结果相当于执行了一次它作为控制码所起的控制效果。显然在行编辑中这个效果正是我们所希望的。

至此，我们就可以在输入正文的当前行时，首先把换行码 CHR(10)放进 Linbuf 的第一个单元中，当该行正文字符输入完毕后，再把回车码 CHR(13)放进该行的最后一个单元，经过上述处理之后，texbuf 中的每一标准行就实现了规范化，即每一行都以换行标志开始而以回车标志结束。当用户欲在终端上显示或在打印机上输出正文内容时，就可以在输出设备上重现用户期望的那一种一行一行的格式而不是唯一的一长行。

1．2 正文缓冲区 texbuf：正文缓冲区的容量可根据用户的需要，兼顾到计算机内存空间大小，系统是单用户还是多用户等诸多因素而定。当 texbuf 设置合适之后，就可以容纳具有首尾标志的各正文行，最后传送 EOT 码以示正文输入结束。

2　行编辑命令与功能设计

不同编辑程序的命令集可以有较大的差异，但具体处理各编辑命令的模式却大致相同。一般情况下，让每条命令对应一个过程，这样可以增加程序的模块化与可读性。

行编程序的命令集可使用情况语句实现之，如:

```
CASE　〈命令集元素〉　OF
    命令 1:　过程 1;
    命令 2:　过程 2;
      :
      :
    命令 n:　过程 n;
    其它:　{跳出 CASE 格式, 读下一命令}
END
```

行编辑命令是用户通过键盘输入的，因此要求尽量的简单明了。当命令被识别后，即转去执行相应的过程，以实现要求的编辑动作。另外，键盘命令与正文内容应保证能被准确的识别。

行编辑命令可以控制在 10 个左右，以尽可能的简化用户的使用手续。其中主要的几个命令可简介如下:

2．1　正文插入命令: 用户逐行的把正文内容依行序读入 texbuf，输入时对每行附加首尾标志。当在原文件中插入若干行或若干字符时，一定要记忆行数与字符数，以便使正文缓冲区内容从当前位置向后顺延相应单元，实现插入。

2．2　行删除命令: 当用户欲删除 texbuf 中的当前行时，由于行尾置有 CR 的标志，故应从当前指示位一直删至 CR 符号处，在此过程中，自动记忆所删字符个数，然后再执行操作，使 texbuf 中相应部分顺序前移等量的存贮单元，不使 texbuf 呈稀疏态。

2．3　texbuf 内容写入磁盘命令: 当 texbuf 已经放满或用户已经完成了对 texbuf 的操作任务时，就需要存入盘，使之成为永久文件。该命令执行两个动作:

2．3．1　打开用户定义的文件名，若磁盘上该文件存在，则显示信息询问:

To delete the file　×××·×××?

这时用户可用 Y／N 响应之。

2．3．2　关闭通道，在盘上形成永久文件。

2．4　由磁盘调文件入 texbuf 命令: 把外设上的文件调入 texbuf，若 texbuf 长度小于文件长度时，显示信息:

Input file too long!

2．5　替换命令: 这是行编辑程序的一个重要功能，它可根据用户键入的被替换串和替换串，记忆两串的长度，并扫描用户希望进行替换动作的行，搜索并记忆该行中所匹配到的子串的起始位置，然后根据下述情况作出相应处理:

2．5．1　两串长度相等。仅执行替换动作: 替换串逐一代替被替换串。

2．5．2　替换串长于被替换串。先用替换串中左起与被替换串等长的部分进行代换，

其余留部分有待于 texbuf 当前位置与尾标志之间内容被依次向后顺延后，腾出与余留部分等长的单元数，然后插入余留部分。由于每行前已假定最长为 80 个字符，故当上述插入后若该行长度超限，应给出相应信息。

2．5．3 情况与上述相反时，则应先作等量代换，然后再把整个正文缓冲区依序前移。

替换命令的示意程序列出如下：

```
    Procedure  subs (y: integer): {替换中的等长部分输入缓冲区}
      var j: integer;
begin
  i:=0
  for j:=p  to  p+y+1  do
   begin
     i:=i+1;
     texbuf[p]:=Linbuf[i];
   end;
  p:=p+y;
end;
Procedure  substitute; {替换命令控制程序}
  Label 10, 20;
  var j: integer;
  begin
    y:=0;
10: read(ch);
  if ch < > '//' then
   begin
      y: y+1;   sud[y]: ch;
      goto 10;
    end;
   m:=p;  k:=1;
  repeat
    LLinbuf[k]:=texbuf[p]; p:=p+1;
    until  texbuf[p]=chr(13);
    g: m+POS(sub, LLinbuf)-1
    p:=g;  x:=0;
20: read(ch);
  if ch < > '//' then
    begin
     x:=x+1;   LLinbuf[x]:=ch;
     goto 20;
    end;
  if  x>y  then
  begin
subs(y);
wtext(x-y); {拉长 texbuf, 插入余留串过程}
for  j:=1  to  x-y  do
texbuf[p+j-1]: =LLinbuf[y+j]
   end
 else
   begin
subs(y);
uptext(y-x); {压缩 texbuf, 删去余留串过程}
     end;
   end;
```

替换命令的格式可设计为：

```
!   S    _____…__/ * * * … * /
     命令   被替换串     替换串
```

还有其它几个编辑命令，如让用户查询各种行编辑命令功能等等，这里不再赘述。

3 行编辑功能的扩充与命令查询

本文对传统的行编辑程序作了功能上的扩充，这主要体现在两个方面：

3．1 一般微型计算机的编辑程序一次只能对一个源文件进行加工，若欲改换加工对象，则必须退出编辑态，然后以另一新文件名作为标识重新进入编辑态方可进行加工。这

么一退一进，无疑在效率上是不高的，尤其当面临更多的加工对象时，问题就愈加明显。

本文所示行编辑程序采用了先进入编辑态然后确立文件标识的处理办法。在这中间可进行编辑所要求的任何加工动作，如插入、删改等等。其特点是这时若需要建立（或调入、删改）别的新（或老）文件时，不需退出编辑态，就可直接进行加工动作。在理论上（只要外存许可）该编辑程序可在编辑态内加工无限多个源文件。显然，它的功能是很强的。

3．2 由于各条编辑命令间都存在着一定的上下文逻辑联系，它实质上体现了用户对完成当前动作后对下一动作的期望，因此我们可以把这些相关的命令束设计成连锁式的，以尽量减轻用户的记忆负担。例如，当前动作为插入正文文体，那么下一个逻辑动作就是存入外设，最后一个动作是进行编译。这几个步骤在通常情况下都是分离进行的，但既然它们之间存在着时间上的连续性，我们就可以把它们融于一条命令之中。当然编译问题已超出了编辑的范畴，但这一步我们可以通过调用一个简单的命令程序（shell）完满的得到实现。考虑到行编辑程序是面向系统中所有高级语言的（解释 BASIC 除外），对不同的语言应设不同的文件词缀以作为调用该语言编译程序的唯一判别字，如·FOR，·PAS，·C，·AS 等。上述处理显然简化和扩充了编辑的功能。我们称这些命令为复合命令，复合命令和与它等功能的单一命令共存于编辑程序中，它们对于用户来说是可选的。它们的存在形式是两张命令表：单一命令表和复合命令表，用户可随时对它们进行查询与显示。上述命令的源程序对用户是透明的，可随时用编辑命令调出阅读。

由于 UNIX 系统提供了各种语言的链接接口，因此我们对某些编辑命令功能的设计就不一定局限于使用某一语言，可以根据各语言的特点，博采众长，设计出效率更高的编辑程序。

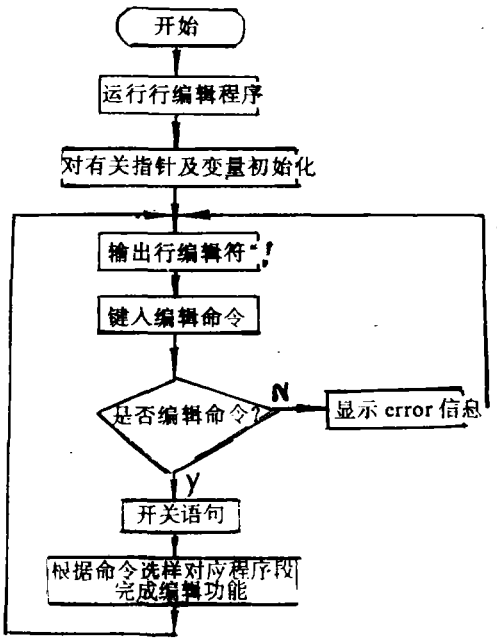
4　后　　记

构造设计一个高效的行编辑程序，考虑的因素较多。但有些基本问题，如怎样进入编辑态，文件系统有何特点，所用的终端特性如何，如何使编辑程序为用户共享等等，都必需首先予以考虑，了解了这些问题后，方可设计所需的数据结构。

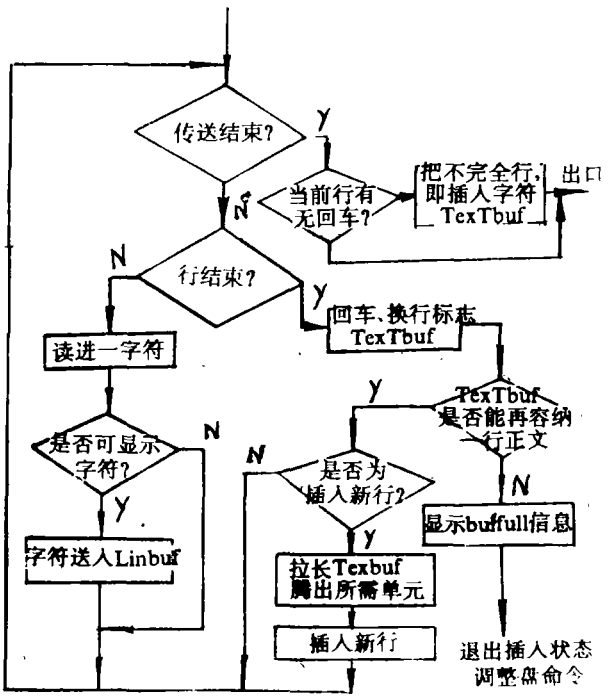
上述工作在手编实现之后，就可尝试利用系统中的软件工具 LEX(Lexical Analysis Generator)去自动生成。这样做有两个优点：节省时间，代码效率高。

从手工编制系统软件到自动生成，这无论在认识上还是在实践上都是一个飞跃，它把人们的思想从一个旧模式中解救出来，这也是今后应用软件设计的必然方向。实现这个飞跃的基础建立在对基本软件结构的精深理解之上。本文的实践就是为了迈出这一步作一些准备工作。

附：行编程序的主控框和正文插入框图简介：



行编主控程序框图



正文插入框图

参 考 文 献

〔1〕 王诚等编. PASCAL程序设计及应用. 清华大学出版社

〔2〕 SVS PASCAL 参考手册(英文)

〔3〕 凌瑞骥等译. 微型计算机软件设计基础. 清华大学出版社

〔4〕 《计算机研究与发展》编辑部. UNIX分时系统程序员手册

The functional expansion and design of the system line editor

Wang Wenyi Sun Lixian

(ZhengZhou Institute of Technology)

Abstract: The author of this article deals with the basic structure of the line editor and the method of its realization in detail. The traditional function of the line editor has been expanded considerably so that the computer's efficiency has been improved, and the machine has become more convenient in use.

Keywords: line editor program, editor command set