Tutorials

Output routines: Examples and techniques Part IV: Horizontal techniques

David Salomon

Note on notation: The logo OTR stands for 'output routine', and MVL, for 'Main Vertical List'.

Abstract. The Output Routines series started in 1990 with three articles. The first is an introduction; the second discusses communications techniques; the third is on insertions. The current article is the result of research efforts for the last three years. It discusses advanced techniques for communicating with the OTR from horizontal mode, making it possible to solve problems that require a detailed knowledge of the contents of the lines of text on the page. Logically, this article should be the third in the series, so new readers are advised to read the first two parts, then this part, and finally the part on insertions.

Also, part II should now be called "Vertical Techniques", instead of "Examples and Techniques".

Introduction

Certain typesetting problems can only be handled by the OTR. Many times, such a problem is solved by *communicating with the* OTR. Ref. 1 discusses the details and shows examples, but here is a short recap. Certain clues (such as a small piece of glue or kern, or a box with small dimensions) are left in the document, normally by means of the primitive `\vadjust` (Ref. 1). The OTR searches `\box255` for clues and, on finding them, modifies the document in the desired way *in the vicinity of the clue.*

Searching `\box255` is done by breaking it up into its components, and checking each to see if it is a clue. A component can be a line of text, interline glue, vertical kern, or anything else that can go into a vertical list. The breakup is done by means of the `\last`*xx* commands.

The problem with this technique is that the clues can only be placed *between* lines of text, and not *inside* a line. We thus say that it is possible to communicate with the OTR from vertical mode, but not from horizontal mode. The reason for this is that a line of text is a box made up of characters of text, and a character is not the same as a box. Specifically, the `\lastbox` command does not recognize a character of text as a box. The following tests are recommended for inexperienced readers:

```
\setbox0=\hbox{ABC}
\unhbox0 \setbox1=\lastbox
\showbox1
\bye
```

```
\setbox0=\hbox{AB\hbox{C}}
\unhbox0 \setbox1=\lastbox
\showbox1
\bye
```

The first test above shows `\box1` to be void, and typesets 'ABC'. In contrast, the second test shows `\box1` to consist of an `\hbox` with the 'C', and typesets only 'AB'. Ref. 1, p. 217 contains a more detailed discussion of this point.

Communicating with the OTR from horizontal mode is, however, very desirable, since many OTR problems can easily be solved this way. Three methods to do this have consequently been developed, and are described here, each followed by an application to a practical problem. Note that each method has its own limitations, and none is completely general.

The main idea in methods 1 and 2 is to enclose each character of text, as it is being read from the input file, in a box. Now there are no longer any characters, just many small boxes. When the OTR is invoked, each line of text is a box containing other boxes (and glue, kern and penalties) but no characters. The `\lastbox` command can now be used to break up the line of text into its components and search for clues. (To simplify the discussion, we assume text without any math, rules, marks or whatsits.)

Before discussing the details of the first two methods, their disadvantages should be mentioned. Since we no longer have any characters, just boxes, we lose hyphenation, kerning and ligatures. As a result, we normally have to use `\raggedright`, so these methods can only be used in cases where a ragged right margin, and lower typesetting quality, are acceptable.

How can we coerce TeX to place each character, as it is being input, in a box? Here are the principles of the first two methods:

Method 1. Declare every character of text active, and define it to be itself in a box. Thus we say '`\catcode'\a=13`' followed by '`\def a{\hbox{a}}`' and repeat for all characters. (The simple definition above cannot be used, because it is infinitely recursive. See below for how it is really done.)

The main disadvantage of this method is that no macros can be embedded in the text. Something like `\abc` will be interpreted as the control sequence '`\a`' followed by the non-letters '`b`' and '`c`'.

Method 2. Use `\everypar` (and also redefine `\par`) to collect an entire paragraph of text in `\toks0`. Scan `\toks0` token by token and place each non-space token in a small `\hbox`. Then typeset the paragraph. This method does not have the disadvantage of the previous one, since there are no active characters.

Method 3 is completely different. It does not place characters in boxes, and does not use `\lastbox` to break up a line of text. Instead it writes `\box255` on a file, item by item, then reads it back, looking for the clues. This method is slow and tedious, but it does not have the disadvantages of the previous two.

Method 1

We need to declare all the letters, digits and punctuations active (actually, I only did this for the lower case letters, for the uppercase 'L', for the three digits '123', and for '.,;'). Each character should now be defined as a box containing its own character code. Turning 'a', e.g., into an active character is done by '`\catcode`\a=13 \def a{\hbox{\char`\a}}`'. When we get to the 'b', however, the command '`\catcode`\b=13 \def b{\hbox{\char`\b}}`' fails because 'a' is no longer a letter, so instead of `\catcode`, TEX sees `\c` followed by a non-letter. The solution is to use '`\let`' to redefine the control sequences `\catcode`, `\def`, `\hbox` and `\char`. Also the number 13 may cause a problem later, after the digit '1' is declared active. The result is declarations such as:

```
\let\?=\catcode \let\!=\active
\let\*=\def \let\+=\char \let\==\hbox
\let\<=\leavevmode \let\\=\bye
```

following which, the active characters can be defined by commands such as '`\?`\a\! *a{\={\+`\a}}`'. After this is done, any character of text input by TEX is expanded into a box containing that character. Note that TEX does not see any text anymore, just a lot of small boxes. This means that there will be nothing to start a paragraph (we will have to place a `\leavevmode` explicitly at the beginning of every paragraph). The following example is a simple application of this technique.

About the examples

All three examples use the following text, that was artificially divided into two paragraphs.

```
in oLden times, when wishing stiLL heLped
one, there Lived a king whose daughters were
aLL beautifuL; and the youngest was so
beautifuL that the sun itseLf, which has
seen so much, was astonished whenever it
shone in her face. cLose by the kings
castLe Lay a great dark forest, and under
an oLd Lime tree

in the forest was a weLL, and when the day
was very warm, the kings chiLd went out
into the forest and sat down by the side of
the cooL fountain;  and when she was bored
she took a goLden baLL, and threw it up on
high and caught it; and this baLL was her
favorite pLaything.
```

Example 1. Widening certain letters

This example uses method 1. Before delving into the details of the example, here is the code used to activate characters and to conduct the test:

```
\hsize=3in\tolerance=7500
\raggedright\zeroToSp

\let\?=\catcode \let\!=\active
\let\*=\def \let\+=\char \let\==\hbox
\let\<=\leavevmode \def\\{\vfill\eject\end}

\?`\a\! \*a{\={\+`\a}}
\?`\b\! \*b{\={\+`\b}}
\?`\c\! \*c{\={\+`\c}}
...
\?`\x\! \*x{\={\+`\x}}
\?`\y\! \*y{\={\+`\y}}
\?`\z\! \*z{\={\+`\z}}
\?`\L\! \*L{\={\+`\L}}
\?`\1\! \*1{\={\+`\1}}
\?`\2\! \*2{\={\+`\2}}
\?`\3\! \*3{\={\+`\3}}
\?`\.\! \*.{\={\+`\.}}
\?`\,\! \*,{\={\+`\,}}
\?`\;\! \*;{\={\+`\;}}

\< in oLden times... Lime tree

\< in the forest... favorite pLaything.
\\
```

The example itself is an interesting OTR problem that has recently been communicated to me (Ref. 2), and was the main reason for developing these OTR methods. If one decides, for some reason, not to hyphenate a certain document, then a ragged right margin is a good choice, which makes the text look best. Certain religious texts, however, don't use hyphenation, and also frown on raggedright. They produce a straight right margin by widening certain letters. In the example below (Figs. 1 & 2) I have selected the 'L', since it's easy to design this letter out of two parts that connect with a rule. I did not actually bother to design a special 'L', and I simply extended it with an `\hrulefill`.

in oLden times, when wishing stiLL
heLped one, there Lived a king whose daughters
were aLL beautifuL; and the youngest was so
beautifuL that the sun itseLf, which has seen so
much, was astonished whenever it shone in her
face. cLose by the kings castLe Lay a great dark
forest, and under an oLd Lime tree
in the forest was a weLL, and when the
day was very warm, the kings chiLd went out
into the forest and sat down by the side of the
cooL fountain; and when she was bored she
took a goLden baLL, and threw it up on high
and caught it; and this baLL was her favorite
pLaything.

Figure 1

in oL___den times, when wishing stiL___L___
heL_ped one, there L_ived a king whose daughters
were aL_L_ beautifuL_; and the youngest was so
beautifuL_ that the sun itseL_f, which has seen so
much, was astonished whenever it shone in her
face. cL_ose by the kings castL_e L_ay a great dark
forest, and under an oL______d L______ime tree
in the forest was a weL__L__, and when the
day was very warm, the kings chiL___d went out
into the forest and sat down by the side of the
cooL______ fountain; and when she was bored she
took a goL_den baL_L_, and threw it up on high
and caught it; and this baL_L_ was her favorite
pL______________________________aything.

Figure 2

When I started thinking about this problem, it was clear to me that this was an OTR problem, and I tentatively outlined the following steps to the solution:

1. Typeset the text with `\raggedright`. This makes the interword glue rigid, and the `\rightskip` glue flexible. Each line of text is placed in an '`\hbox to \hsize`', and `\rightskip` is stretched as necessary.

2. In the OTR, break `\box255` up into individual lines of text. For each line, perform steps 3 through 6.

3. Perform an `\unhbox` on the line, to return `\rightskip` to its natural size (zero). Subtract the present width of the line from its original width (`\hsize`). The difference is the amount by which all the L's on the line will have to be stretched.

4. Break the line up into individual components (mostly characters, glue, and penalties), and count the number of L's in the line.

5. Divide the difference from step 3 by the number of L's from step 4. The result is the amount by which each L will have to be widened.

6. Break the line up again, widening each L. Pack the line in a new `\hbox`.

7. Rebuild the page from the line boxes generated in 6, and ship it out.

The only problem was step 4. A line of text cannot normally be broken up into individual characters and examined. However, using method 1, it is possible to break up such a line, since it does not include any characters, and search for clues. A clue, in our case, is a box whose width is the same as that of an 'L' (if other characters happen to have the same width, the width of the 'L' can be changed by 1sp).

The seven steps above can now be implemented, using the breakup technique (Ref. 1, p. 214).

Step 1. Just say `\raggedright`.

Steps 2 and 7. The OTR becomes

```
\newbox\brk
\output={\setbox\finPage=\vbox{}%
 \setbox\brk=\vbox{\unvbox255 \breakup}%
 \ifdim\ht\brk>0pt
  \message{Incomplete breakup,
           \the\ht\brk}\fi
 \shipout\box\finPage \advancepageno}

\newif\ifAnyleft \newcount\pen
\newbox\finPage
\def\breakup{%
 \loop \Anyleftfalse
  \ifdim\lastskip=0pt
  \else \Anylefttrue
   \skip0=\lastskip \unskip
   \global\setbox\finPage
     =\vbox{\vskip\skip0 \unvbox\finPage}%
  \fi
  \ifdim\lastkern=0pt
  \else \Anylefttrue
   \dimen0=\lastkern \unkern
   \global\setbox\finPage
```

```
      =\vbox{\kern\dimen0 \unvbox\finPage}%
  \fi
  \ifnum\lastpenalty=0
  \else\Anylefttrue
   \pen=\lastpenalty \unpenalty
   \global\setbox\finPage
     =\vbox{\penalty\pen \unvbox\finPage}%
  \fi
  \setbox0=\lastbox
  \ifvoid0 \else
    \Anylefttrue\message{.}\breakupline
    \global\setbox\finPage
      =\vbox{\box2 \unvbox\finPage}%
  \fi
  \ifAnyleft
 \repeat}
```

Macro `\breakup` is essentially the same as in Ref. 1. It places each line of text in `\box0`, and expands `\breakupline`. Note the lines with `\global\setbox\finPage=...` They rebuild the page, line by line, in `\box\finPage` (step 7). When the entire process is complete, the OTR ships out `\box\finPage`.

Steps 3 and 5. Macro `\breakupline` resets the line of text to its natural width, calculates the width difference in `\diff`, expands `\countLonline` to count the number of L's in the line, divides `\diff` by that number, and expands `\longLline` to actually widen the L's in the line.

```
\newdimen\diff \newcount\Lnum
\def\breakupline{\diff=\hsize
 \setbox0=\hbox{\unhbox0}
 \advance\diff-\wd0
 \Lnum=0
 \setbox1=\hbox{\unhcopy0 \countLonline}
 \ifdim\wd1>0pt \message{%
   Incomplete line breakup}\fi
 \ifnum\Lnum=0 \diff=0pt
 \else \divide\diff by\Lnum
 \fi
 \setbox2=\null
 \setbox1=\hbox{\unhbox0 \longLline}}
```

Step 4. Macro `\countLonline` breaks the line up into individual components and counts the number of L's in the line. It uses a breakup loop similar to the one in `\breakup` above. Note how boxes with 'L' are identified by their width.

```
\newif\ifCharleft
\def\countLonline{%
  \Charleftfalse
  \ifdim\lastskip=0pt\else \Charlefttrue
   \skip0=\lastskip \unskip\fi
  \ifdim\lastkern=0pt\else \Charlefttrue
   \dimen0=\lastkern \unkern\fi
  \ifnum\lastpenalty=0 \else\Charlefttrue
   \pen=\lastpenalty \unpenalty\fi
  \setbox1=\lastbox
  \ifvoid1\else
    \ifdim\wd1=6.25002pt
      \global\advance\Lnum1
    \fi
    \Charlefttrue
  \fi
  \ifCharleft \countLonline\fi}
```

Step 6. Macro `\longLline` uses the same technique to break the line up again, extend all the 'L's, and rebuild it in `\box2`. Macro `\extendL` packs an 'L' with an `\hrulefill` in a new `\hbox`.

```
\newif\ifSomeleft
\def\longLline{%
  \Someleftfalse
  \ifdim\lastskip=0pt\else \Somelefttrue
   \skip0=\lastskip \unskip \global\setbox2
     =\hbox{\hskip\skip0 \unhbox2}\fi
  \ifdim\lastkern=0pt\else \Somelefttrue
   \dimen0=\lastkern \unkern \global\setbox2
     =\hbox{\kern\dimen0 \unhbox2}\fi
  \ifnum\lastpenalty=0 \else\Somelefttrue
   \pen=\lastpenalty \unpenalty \global
    \setbox2=\hbox{\penalty\pen \unhbox2}\fi
  \setbox1=\lastbox
  \ifvoid1\else
    \ifdim\wd1=6.25002pt \extendL\fi
    \setbox2=\hbox{\box1 \unhbox2}%
    \global\Somelefttrue
  \fi
  \ifSomeleft \longLline\fi}

\newdimen\Lwidth
\def\extendL{%
  \Lwidth=\wd1 \advance\Lwidth by\diff
  \setbox1=
    \hbox to\Lwidth{\unhbox1\hrulefill}}
```

The code is somewhat long, but is well structured, and most macros use the same breakup technique.

Problems. 1. A line of text without L's is not extended, so it normally comes out shorter.

2. Since there are no letters in our texts, just boxes, there is nothing to signify the start of a paragraph. Each paragraph must therefore start with a `\leavevmode` command (`\<` in our case).

3. `\box255` may only contain boxes, glue, kern and penalties. Anything else (such as text, rules, whatsits or marks) would stop the breakup macros.

Note that overfull lines contain rules, so they should be avoided (by increasing the tolerance, increasing the stretch of \rightskip, or by rewriting the text).

4. Because of reasons discussed in Ref. 1, glues with a natural size of 0pt stop the breakup macros. Macro \zeroToSp below changes the natural size of several such glues to 1sp. It also changes the plain values of some common penalties from 0 to 1. This macro should be expanded once, at the start of the document.

```
\def\zeroToSp{\parskip=1sp plus1pt
  \parfillskip=1sp plus1fil
  \advance\leftskip by1sp
  \advance\rightskip by1sp
  \def\vfil{\vskip1sp plus1fil} %
  \def\vfill{\vskip1sp plus1fill}%
  \abovedisplayshortskip=1sp plus3pt
  \postdisplaypenalty=1
  \interlinepenalty=1}
```

5. To identify boxes with an 'L', we use the width of an 'L' in font cmr10. To guarantee reliable identification, no other character in the font should have the same width.

Possible improvements and applications. 1. If \diff is less than \hfuzz (or some other small parameter) it can be set to zero, since there is no point in widening a letter by a very small amount.

2. The L's on the last line of a paragraph are normally widened a lot. If this is not desirable, the macros can be changed to treat the last line differently.

Method 2

As mentioned earlier, the principle is to collect the text of an entire paragraph in a toks register, then to scan the register token by token, placing each character token in a small \hbox. We again lose hyphenation, kerning and ligatures, so we normally have to resort to a ragged right margin. However, we can have control sequences embedded in the text. Care should be taken to identify each control sequence (and its argument) and to expand it, instead of placing it in a box. Here are the macros and the test text:*

```
\hsize=4in \tolerance=7500
\raggedright \zeroToSp
\begingroup
\newif\ifargmn
\everypar{\catcode` =12 \toks0=\bgroup}
\def\par{\catcode` =10 \argmnfalse
 \expandafter\Tmp\the\toks0 \end \endgraf}
\def\Tmp#1{\ifx\end#1\def\next{\relax}%
 \else
 \ifargmn\cs{#1}\argmnfalse
 \else
 \ifcat\relax\noexpand#1%
   \let\cs=#1\argmntrue
 \else
 \ifnum11=\catcode`#1\hbox{#1}%
 \else
 \ifnum12=\catcode`#1\hbox{#1}%
 \else\ifnum'40=`#1\ \fi
 \fi
  \fi
   \fi
    \fi
 \let\next=\Tmp\fi\next}%
%
in\Mnote{xyz *} oLden times, when
\Mnote{abc 2}wishing stiLL
heLped\Mnote{note 3} one, there Lived a
king whose daugh\Mnote{note 4}ters
were aLL beautifuL; and the
youngest\Mnote{note 5} was so beautifuL that
the sun itseLf, which has seen so much, was
\Mnote{note 6}astonished whenever it
shone in her face. cLose by the kings
castLe L\Mnote{note 7}ay a great dark
forest, and under an oLd Lime tree}

in the fore\Mnote{note 8}st was a weLL,
and when the day was\Mnote{note 9} very
warm, the kings chiLd went out into the
fores\Mnote{note 10}t and sat down by the
side of the coo\Mnote{note 20}L fountain;
\Mnote{note 21}and when she was bored
\Mnote{note 22}she took a goLden
baLL,\Mnote{note 12} and threw it up on high
and caught it; and this baLL was her favorite
\Mnote{note 13}pLaything.}

\endgroup
\bye
```

* Editor's note: This text, used to produce Figures 3 and 4, has been realigned to fit the narrow *TUGboat* measure.

The \everypar parameter is modified to place '\toks0=\bgroup' at the start of each paragraph. At the end of a paragraph we need a closing \egroup, which is easy to insert by redefining \par. Unfortunately, the command '\toks0=\bgroup...\egroup' does not work. Using \bgroup is okay, but a right brace (a token of catcode 2) is required, instead of the control sequence \egroup. When using this method we unfortunately have to insert a '}' explicitly at the end of every paragraph. This is one of two unsolved problems with this method.

The `\par` primitive is modified to expand '`\the\toks0`', to append an `\end` to it, to expand `\Tmp`, and to close the paragraph.

Macro `\Tmp` uses recursion to extract the next token from `\toks0` and to test it. Tokens with catcodes 11 and 12 are placed in boxes and appended to the current list (normally the MVL) (except spaces, which are appended as spaces to the MVL). Control sequence tokens are also identified. Each such token is kept in `\cs` until its argument is identified in the following recursive iteration, where it is expanded. In the current version, any control sequence embedded in the text must have exactly one argument. The changes of `\everypar` and `\par` are confined to a group.

Spaces present a special problem. The scanning of tokens skips all spaces. Therefore, the catcode of space had to be changed. It has been changed to 12 (other) and `\Tmp` identifies spaces by their character code. When a catcode 12 space is identified by `\Tmp`, a normal (catcode 10) space is appended to `\box0`.

The second unsolved problem in this method is the end of lines. They are converted into spaces, but only after the catcode of a space has been changed. As a result, they appear in `\toks0` as normal spaces (catcode 10) and are skipped.

Example 2. Marginal notes

Typesetting notes in the margins of a scholarly book is very common. Ref. 3 is an interesting example, familiar to many TEX users. Another well known example is the marginal notes of the mathematician Pierre Fermat. When trying to prove the so-called Fermat's last theorem (there is no integer $n > 2$ such that $x^n + y^n = a^n$ for rational x, y and a), he wrote in the margin of the book he was reading (Bachet's *Diophantus*) "I have discovered a truly marvellous demonstration of this general theorem, which this margin is too narrow to contain" (Ref. 4). Unfortunately for us, to this day no one has been able to prove (or find a counterexample to) this theorem.* I like to call this famous note *Fermat's warning*. It warns us not to abuse this useful tool of the author.

* Editor's note: While this article was in production, it was announced that Andrew Wiles of Princeton University had found a proof, then that a gap may exist in the proof; Wiles is continuing work on the paper.

When teaching TEX I have always noticed how, when discussing marginal notes, the class suddenly comes to life and starts following the discussion with renewed interest. In the lab that follows, people start writing macros for marginal notes, invariably ignoring Fermat's warning, and overdoing this useful feature.

A single note can easily be placed in the margin of a given line with the help of `\vadjust`. When writing a text with many marginal notes, however, the writer may end up with two or more notes appearing on the margin of the same line. Because of the limited space on the margin, the notes for the same line of text may have to be rearranged before the page is shipped out, and this is an OTR problem. Rearranging notes may involve placing some on the left, and some on the right margin; it may mean to typeset them in very small type, to move some up or down (if there is room on adjacent lines), or to warn the author that there is no room.

In this example, rearranging is done in a simple way. The first note found on a line is typeset on the left, the second one, on the right margin. If more notes are found on the same line, none is typeset, and a warning, with the input line number, is placed in the log file.

The implementation is straightforward. Macro `\Mnote` places the text of the note in an '`\hbox to1sp`' inside the paragraph '`\def\Mnote#1{\hbox to1sp{#1\hss}}`'. Method 2 is used to place every character of text in a box. The OTR breaks up `\box255` into its top level components and identifies the lines of text. Each line is further broken up, and all the clues (boxes of width 1sp) in it located. Depending on how many clues were found, the macros place the notes as described above. The OTR is straightforward:

```
\newbox\brk
\output={\setbox\finPage=\vbox{}
  \setbox\brk=\vbox{\unvcopy255 \breakup}%
  \ifdim\ht\brk>0pt \message{Incomplete
    breakup, \the\ht\brk}\fi
  \shipout\box255 \shipout\box\finPage}
```

Note that it also ships out `\box255`, for comparison purposes. Macro `\breakup` rebuilds all the elements of `\box255` in `\box\finPage`, except that each line of text is further broken up by `\breakupline` (and the notes properly placed in the margins) before being rebuilt and appended to `\box\finPage`.

```
\newif\ifAnyleft \newcount\pen
\newbox\finPage
\def\breakup{%
 \loop \Anyleftfalse
```

```
  \ifdim\lastskip=0pt
  \else
    \Anylefttrue \skip0=\lastskip \unskip
    \global\setbox\finPage
      =\vbox{\vskip\skip0 \unvbox\finPage}%
  \fi
  \ifdim\lastkern=0pt
  \else
    \Anylefttrue \dimen0=\lastkern \unkern
    \global\setbox\finPage
      =\vbox{\kern\dimen0 \unvbox\finPage}%
  \fi
  \ifnum\lastpenalty=0
  \else \Anylefttrue
    \pen=\lastpenalty \unpenalty
    \global\setbox\finPage
      =\vbox{\penalty\pen \unvbox\finPage}%
  \fi
  \setbox0=\lastbox
  \ifvoid0
  \else \Anylefttrue\message{.}%
    \breakupline
    \global\setbox\finPage
      =\vbox{\box2 \unvbox\finPage}%
  \fi
  \ifAnyleft
 \repeat}
```

Macro `\breakupline` expands `\countNotesonline` to break up one line of text, and count the number of notes. It then rebuilds the line in `\box2` with the notes placed in the margins, and with the special boxes emptied.

```
\newcount\numnotes
\def\breakupline{\numnotes=0
 \setbox1=\hbox{\unhbox0 \countNotesonline}%
 \ifdim\wd1>0pt \message{%
   Incomplete line breakup}\fi
 \ifcase\numnotes
 \relax % \numnotes=0 -> 0 notes on this line
 \or    % 1 note
  \setbox2=\hbox to\hsize{%
   \llap{\box3\kern3pt}\unhbox2\hfil}%
 \else  % 2 or more notes
  \setbox2=\hbox to\hsize{%
   \llap{\box4\kern3pt}\unhbox2\hfil
   \rlap{\kern3pt\box3}}%
 \fi}
```

Macro `\countNotesonline` is a simple application of the breakup technique for one line of text. The first note found in the line is placed in `\box3`, and the second one, in `\box4`. All subsequent notes are flushed. A small dash is inserted in each special box to show where the note came from.

```
\newif\ifCharleft
\def\countNotesonline{%
  \Charleftfalse
  \ifdim\lastskip=0pt
  \else \Charlefttrue
    \skip0=\lastskip \unskip
    \global\setbox2
      =\hbox{\hskip\skip0 \unhbox2}%
  \fi
  \ifdim\lastkern=0pt
  \else \Charlefttrue
    \dimen0=\lastkern \unkern
    \global\setbox2
      =\hbox{\kern\dimen0 \unhbox2}%
  \fi
  \ifnum\lastpenalty=0
  \else \Charlefttrue
    \pen=\lastpenalty \unpenalty
    \global\setbox2=
      \hbox{\penalty\pen \unhbox2}%
  \fi
  \setbox1=\lastbox
  \ifvoid1\else
   \ifdim\wd1=1sp % a special box
     \ifnum\numnotes=0
       \global\setbox3=\hbox{\unhbox1}%
     \fi
     \ifnum\numnotes=1
       \global\setbox4=\hbox{\unhbox1}%
     \fi
     \ifnum\numnotes>1
       \global\setbox3=\hbox{!!!}%
       \global\setbox4=\hbox{!!!}%
       \message{Too many notes on line
           \the\inputlineno}%
     \fi
     \global\advance\numnotes 1
     \global\setbox2=
       \hbox{\pop\unhbox2}%
   \else % not a special box
   \global\setbox2=\hbox{\box1 \unhbox2}%
   \fi
   \Charlefttrue
  \fi
  \ifCharleft \countNotesonline\fi}
```

Finally, macro `\pop` places a small dash in the special box after it has been emptied. The dash is character "37 of font cmsy (the math symbols). This character is constructed in a box of width 0 and it sticks out on the right. Normally it is followed by a minus or a right arrow, to create a "maps to" symbol (Ref. 5, p. 515).

```
\def\pop{\leavevmode\raise4pt\hbox to1sp
 {\hss\smash{\tensy\char"37}\kern1.2pt\hss}}
```

Tests

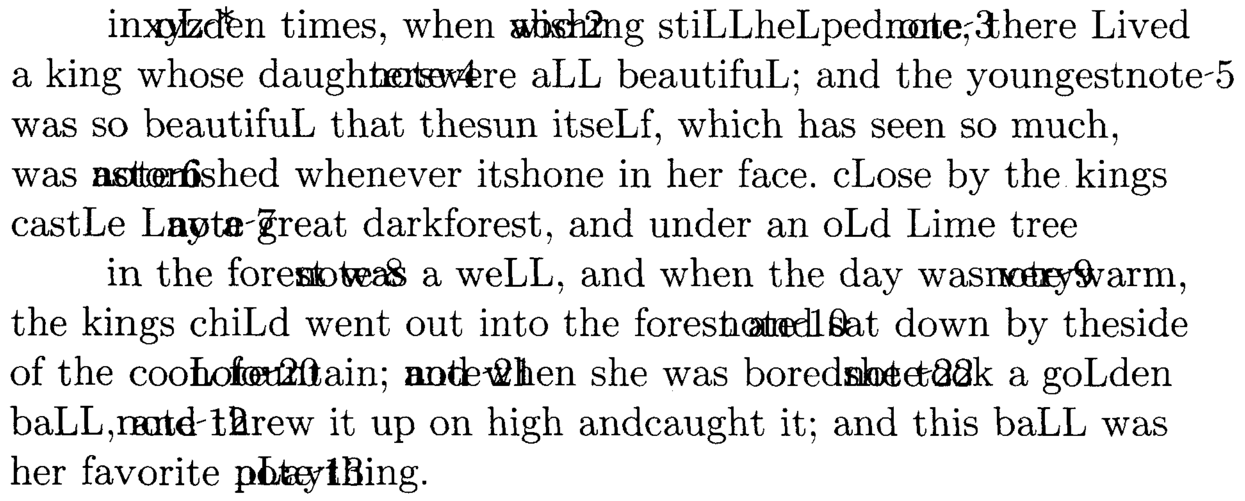
The two paragraphs used for the test were shown earlier. The first diagram (Fig. 3) shows \box255 before any changes. Note how the text of the notes overlap the text of the paragraphs, since they are saved in boxes *inside the paragraph*. The diagram in Fig. 4 shows the final result shipped out.

In practical use, sophisticated macros can be developed that will set the notes in small type, will number them consecutively, and will move them vertically, if necessary. However, as long as they are based on the principles shown here, raggedright will normally have to be used, which is not always acceptable.

Method 3

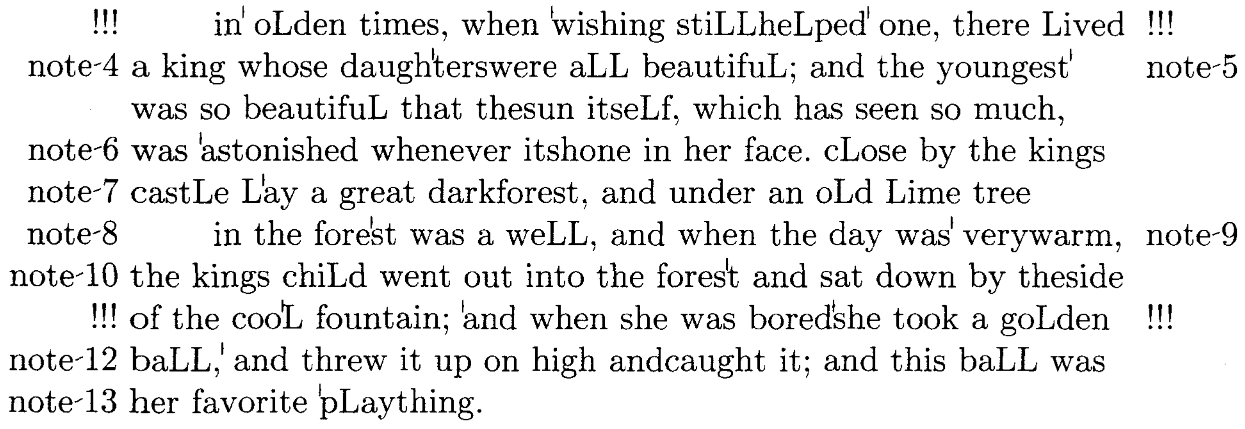
This method is based on a two-pass job. In the first pass the text is typeset in the normal way, with characters, not boxes. Clues are inserted in the text, to be found later, by the OTR, in pass 2. Pages can either be shipped out or trashed, but the OTR writes \box255 on a file, to be read by pass 2. Advanced users know that a box cannot be written on a file in the usual way, using \write. The novelty of this method is that a box can be written on the *log file*, using \showbox.

The user has to make sure that the log file is saved after pass 1. Pass 2 reads the contents of \box255 from the file, searches for the beginning of each line of text, and for clues inside the line.



```
    in[illegible]en times, when [illegible]ng stiLLheLped[illegible]here Lived
a king whose daugh[illegible]re aLL beautifuL; and the youngestnote-5
was so beautifuL that thesun itseLf, which has seen so much,
was [illegible]shed whenever itshone in her face. cLose by the kings
castLe L[illegible]reat darkforest, and under an oLd Lime tree
    in the fore[illegible] a weLL, and when the day was[illegible]arm,
the kings chiLd went out into the fores[illegible]t down by theside
of the coo[illegible]ain; [illegible]en she was bored[illegible]k a goLden
baLL,[illegible]rew it up on high andcaught it; and this baLL was
her favorite [illegible]ing.
```

Figure 3



```
     !!!     in oLden times, when wishing stiLLheLped one, there Lived  !!!
  note-4 a king whose daughterswere aLL beautifuL; and the youngest     note-5
         was so beautifuL that thesun itseLf, which has seen so much,
  note-6 was astonished whenever itshone in her face. cLose by the kings
  note-7 castLe Lay a great darkforest, and under an oLd Lime tree
  note-8     in the forest was a weLL, and when the day was verywarm,  note-9
 note-10 the kings chiLd went out into the forest and sat down by theside
     !!! of the cooL fountain; and when she was boredshe took a goLden  !!!
 note-12 baLL, and threw it up on high andcaught it; and this baLL was
 note-13 her favorite pLaything.
```

Figure 4

If successful, pass 2 knows what clues are stored in each text line. Pass 2 then reads the source file, typesets it in the usual way and has the OTR modify `\box255`, before shipping it out, according to the clues read earlier.

Note that `\lastbox` is not used. The details of each line of text in `\box255` are read from the file. The main advantage of this approach is that none of the high quality typesetting features, such as hyphenation, kerning and ligatures, is lost.

The main problem with this approach is how to read and analyse the contents of `\box255` from the log file in pass 2 (an example of such a file is shown below for the benefit of inexperienced readers). This turns out to be easy, and it involves the following tasks:

1. Certain records contain backslashes that should be ignored. Examples are: '`..\tenrm i`', '`.\glue(\topskip) 3.05556`' and '`..\glue 3.33333 plus 1.66666 minus 1.11111`'. To ignore these, pass 2 uses the following declarations (inside a group):

```
\let\vbox=\relax \let\glue=\relax
\let\topskip=\relax \let\kern=\relax
\let\rightskip=\relax
\let\baselineskip=\relax
\let\parfillskip=\relax
\let\parskip=\relax
\def\shipout\box{\bgroup}%
\let\showbox=\egroup
\let\discretionary=\relax
```

2. Other records are important and should be identified. Examples are:

a. '`> \box255=`' (this signals the start of the box)

b. '`.\hbox(6.94444+1.94444)x216.81, glue set 0.45114`' (this signals a new line of text).

b. '`..\hbox(0.0+0.0)x0.00002, glue set ...`' (this is a box of width 1sp, denoted a clue of type 1).

c. '`! OK (see the transcript file).`' (this signals the end of the box).

Records of type a are identified by defining `\def\box255={\global\clues={(}}`. The definition of `\box255` is changed (locally) to insert a '`(`' in the toks register `\clues`.

Records of type b are identified by redefining `\hbox`.

```
\def\hbox(#1)x#2 {\toks0={}\one#2\end
  \tmp=\the\toks0 pt
  \ifnum\tmp=1sp\appendclue1
  \else
   \ifnum\tmp=2sp\appendclue2
   \else
    \ifnum\tmp=\Hsize\appendclue+
  \fi\fi\fi}
\def\one#1{\def\arg{#1}%
  \ifx\end#1\let\rep=\relax
  \else\ifx\comma\arg\let\rep=\one
   \else\toks0=\expandafter{\the\toks0 #1}%
     \let\rep=\one
  \fi\fi\rep}
\def\appendclue#1{\global\clues=%
  \expandafter{\expandafter#1\the\clues}}
```

Parameter '`#2`' is the width of the `\hbox`. In the records that interest us, it is either `\hsize` or 1sp or 2sp. The examples in b above show that the width is followed by a comma and a space, but there are records on the log file (such as the paragraph indentation '`..\hbox(0.0+0.0)x20.0`') where the width is followed by a space. This is why '`#2`' in the definition of `\hbox` is delimited by a space. If the width is followed by a comma it (the comma) is removed by macro `\one`. The width is stored in the `\dimen` register `\tmp`.

Macro `\hsize` thus identifies the important records, and appends the tokens '`+`', '`1`' or '`2`' to the toks register `\clues` every time a line of text, or a clue of type 1 or type 2, respectively, is found.

The end of the box in the log file is identified when a type c record is found. We simply compare each record read to the string '`! OK (see the transcript file). `'. When finding it, a '`(`' is appended to `\clues`, and the loop reading the file is stopped. Note that our macros are supposed to stop reading when the end of box is found. They are never supposed to read the end of file. If an end of file is sensed while reading the log file, an error must have occurred.

All the clues found in the log file for one page (a single `\box255`) are stored in the toks register `\clues`, so that later macros can easily find out what clues were found in what text lines. A simple example is the tokens '`)21++2+++121+(`' where the '`)`' and '`(`' stand, respectively, for the end and start of `\box255` in the log file, each '`+`' stands for a line of text, and each '`1`' or '`2`', for a clue of type 1 or 2 found in that line. Thus in the above example, a type 2 followed by a type 1 clue were found in the bottom line, another type 2 clue, in line 3 from the bottom, and three more clues, in line 6 (the top line).

Pass 1 normally writes several boxes on the log file, each corresponding to a page. The following appears in the log file between pages, and has to be 'neutralized'.

```
<output> {\showbox 255
                              \shipout \box 255}
```

This is done by the weird definitions '`\def\shipout\box{\bgroup}`' and '`\let\showbox=\egroup`'. The method is illustrated below by applying it to a practical example.

Example 3. Revision bars

Certain documents, such as the bylaws of an organization, go through periodic revisions. It is good practice to typeset each new revision with vertical bars on the left of parts that have been revised. This is an OTR problem (note that a revision may be broken across pages), and the solution shown here requires the two passes mentioned above. Pass 1 involves:

1. Two macros are defined, to indicate the start and end of each revision.

```
\def\({\leavevmode\raise4pt\hbox to1sp{%
  \hss\smash{\tensy\char"37}\kern1.2pt\hss}}
\def\){\leavevmode\raise4pt\hbox to2sp{%
  \hss\smash{\tensy\char"37}\kern1.2pt\hss}}
```

The macros also place small dashes in the text, to indicate the boundaries of the revision.

2. The OTR writes `\box255` on the log file, and can also ship it out, for later comparison. If a shipout is not required, the OTR can say `\box255=\null \deadcycles=0` instead of `\shipout\box255`.

```
\hsize=3in \vsize=2.2in \tolerance=7500
\showboxbreadth=1000 \showboxdepth=10
\output={\showbox255 \shipout\box255
 \advancepageno}

\input source
\vfill\eject
```

The log file is saved between the passes. Note that the two passes can be parts of the same TeX job, and the log file can be saved when TeX stops, as usual, for a user's response, after the `\showbox`. Pass 2 starts by opening the log file, if it exists:

```
\newread\logfile
\newtoks\clues \newdimen\tmp
\newif\ifmore \moretrue
\newdimen\Hsize \Hsize=\hsize
\newbox\brk \newbox\bars
\newif\ifendRev  \newif\ifbegRev
\newif\ifRev \newif\ifSplitrev

\immediate\openin\logfile=Log
\ifeof\logfile\errmessage{No log file}\fi
```

Note that the file name 'Log' is used here. In the general case, it is possible to read the name from the keyboard. Now comes the OTR. It is divided into two phases. Phase 1 reads a chunk off the log file, corresponding to one page, and prepares tokens in `\clues`. Phase 2 starts the breakup of `\box255`, and ships out `\box\bars` (stretched to `\vsize`) and `\box255`, side by side.

```
\output={%
% Phase 1. Read a chunk off the log file
% and prepare codes in \clues
 \begingroup
 \def\appendclue#1{\global\clues=%
  \expandafter{\expandafter#1\the\clues}}
 \def\OK{! OK (see the transcript file). }
 \def\comma{,}
 \def\box255={\global\clues={()}
 \def\hbox(#1)x#2 {\toks0={}\one#2\end
   \tmp=\the\toks0 pt
   \ifnum\tmp=1sp\appendclue1
   \else \ifnum\tmp=2sp\appendclue2
    \else \ifnum\tmp=\Hsize\appendclue+
   \fi\fi\fi}
 \def\one#1{\def\arg{#1}%
   \ifx\end#1\let\rep=\relax
   \else\ifx\comma\arg\let\rep=\one
    \else\toks0=\expandafter{\the\toks0 #1}%
      \let\rep=\one
   \fi\fi\rep}
%
 \let\vbox=\relax \let\glue=\relax
 \let\topskip=\relax \let\kern=\relax
 \let\rightskip=\relax
 \let\baselineskip=\relax
 \let\parfillskip=\relax
 \let\parskip=\relax
 \def\shipout\box{\bgroup}
 \let\showbox=\egroup
 \let\discretionary=\relax
 \setbox0=\vtop{\hsize=\maxdimen
  \loop
    \read\logfile to\rec
    \ifeof\logfile\morefalse
      \message{end of log file!}
    \else
     \ifx\rec\OK\appendclue)\morefalse\fi
     \rec
    \fi
    \ifmore
   \repeat}
 \endgroup
 \nextclue
 \if)\clue \else\message{Bad clue}\fi
```

```
% Phase 2.
% Breakup \box255 and use the clues
 \global\setbox\bars=\vbox{}%
 \global\endRevfalse \global\begRevfalse
 \global\Revfalse
 \setbox\brk=\vbox{\unvcopy255 \breakup}%
 \ifdim\ht\brk>0pt \message{%
   Incomplete breakup, \the\ht\brk}\fi
 \shipout\hbox{\vbox to\vsize{\unvbox\bars}%
   \kern4pt\box255} \advancepageno}
```

3. Macro `\breakup` breaks up `\copy255` into its top level components. For each component with a dimension, the macro places either a skip or a vrule in `\box\bars`. It is important to realize that when we say, e.g., `\skip0=\lastskip` we lose the specific glue set ratio of `\box255`. This is why the rules are placed in `\box\bars` using `\leaders` and not `\vrule`. This way `\box\bars` can later be stretched to `\vsize`, and all the leaders in it will be stretched.

Exercise: Why is it that a rule placed by means of `\vrule height\skip0` cannot be stretched later?

Answer: Because the command `\vrule` is supposed to be followed by a `height<dimen>`. If we use glue, such as `\skip0`, only the natural size is used, and the stretch and shrink components are ignored.

```
\newif\ifAnyleft \newcount\pen
\def\breakup{%
 \loop \Anyleftfalse
  \ifdim\lastskip=0pt
  \else \Anylefttrue
   \skip0=\lastskip \unskip
   \global\setbox\bars=\vbox{\ifRev\leaders
     \vrule\fi\vskip\skip0\unvbox\bars}%
  \fi
  \ifdim\lastkern=0pt
  \else \Anylefttrue
   \dimen0=\lastkern \unkern
   \global\setbox\bars=\vbox{\ifRev\leaders
     \vrule\fi\kern\dimen0\unvbox\bars}%
  \fi
  \ifnum\lastpenalty=0
  \else\Anylefttrue
    \pen=\lastpenalty \unpenalty
  \fi
  \setbox0=\lastbox
  \ifvoid0 \else \Anylefttrue
   \dimen0=\ht0 \advance\dimen0 by\dp0
   \setbox2=\vbox{\unhbox0 \searchclues}%
   \ifbegRev
   \ifendRev
%TT
     \global\Revfalse \global\endRevfalse
     \global\setbox\bars=\vbox{\leaders
       \vrule\vskip\dimen0\unvbox\bars}%
   \else
%TF
     \global\Revfalse
     \ifSplitrev \global\Splitrevfalse
       \global\setbox\bars=\vbox{\leaders
         \vrule\vskip\ht\bars}%
       \global\setbox\bars=\vbox{\leaders
         \vrule\vskip\dimen0\unvbox\bars}%
     \else
     \global\setbox\bars
       =\vbox{\vskip\dimen0\unvbox\bars}%
   \fi\fi
  \else
   \ifendRev
%FT
    \global\Revtrue
    \global\setbox\bars=\vbox{\leaders
     \vrule\vskip\dimen0\unvbox\bars}%
   \else
%FF
    \global\Revfalse
    \global\setbox\bars
      =\vbox{\vskip\dimen0\unvbox\bars}
  \fi\fi\fi
  \ifAnyleft
 \repeat}
```

When a line of text is found, `\searchclues` is expanded (see below), to update variables `\begRev` and `\endRev`. Four cases are possible:

a. Both variables are false (case FF above). This means no revisions have been found yet. A skip, equal in height to the current line of text, is appended to `\box\bars`. Variable `\Rev` is set to false, indicating that any future components found in `\box255` should become skips in `\box\bars`.

b. `\begRev` is false and `\endRev` is true (case FT above), meaning the current line contains the end of a revision. A rule, the height of the current line, is appended to `\box\bars`. Also, `\Rev` is set to true, indicating that any future components found in `\box255` should become rules in `\box\bars`.

c. Case TT. A revision starts on this line. A rule is appended to `\box\bars` but `\Rev` is set to false. (Also `\endRev` is set to false, so case TF will be in effect from now on.)

d. Case TF. Normally this indicates a line with no revisions but, if `\Splitrev` is true, we have just found the start of a revision that will end on the next page. In this case, `\box\bars` (which has only skips in it so far) is filled up with a rule.

Macro `\searchclues` removes the next token from `\clues` and, if it is 1 or 2, sets `\begRev` or `\endRev` to true, respectively. Note that a revision may start and end on the same line. If the start of a revision is found while `\endRev` is false, it means that the revision will end on the next page. In such a case, variable `\Splitrev` is set to true, indicating that the entire `\box\bars` should be filled with a rule.

```
\def\searchclues{\nextclue
  \if+\clue\let\Next=\relax
  \else
    \if(\clue\let\Next=\relax
      \message{bad Clue}
    \else
      \if2\clue \global\endRevtrue
        \global\begRevfalse
        \let\Next=\searchclues
      \else
       \if1\clue \global\begRevtrue
         \let\Next=\searchclues
         \ifendRev
         \else\global\Splitrevtrue\fi
       \else
         \message{bad clue}
  \fi\fi\fi\fi\Next}

\def\nextclue{\expandafter\extr\the\clues X}
\def\extr#1#2X{\gdef\clue{#1}%
  \global\clues=\expandafter{#2}}
```

The rest of pass 2 is straightforward.

```
\zeroToSp
\input source
\bye
```

For a multi-page document, the OTR performs the same tasks for each page. It first receives `\box255` of page 1. It reads the corresponding lines from the log file, looking for clues and storing them in `\clues`. The OTR then breaks `\box255` up, isolating the lines of text from the bottom. It uses the tokens in `\clues` to modify only the right lines. At the end, `\box255` (and `\box\bars`) are shipped out. The process repeats for each successive page sent to the OTR.

For each page, the OTR reads another chunk off the log file. This is why the two passes must typeset the same text. The best way to handle this is to `\input` the text in the two passes from the same source file. It is possible to make the macros more robust by checking to see, in pass 2, that the chunk read from the log file actually has the same number of text lines as the current `\box255`.

The source file for our test is, as usual:

```
in oLden times, when wishing stiLL heLped
one, there Lived a king whose daughters
were aLL beautifuL; and th\(e youngest
was so beautifuL that the sun itseLf,
which has seen so much, was astonished
whenever it shone in her face.
cLose by the kings castLe Lay a great dark
forest, and under an oLd Lime tree

in the forest\) was a weLL, and when the
day was very warm, the kings chiLd went
out into the forest and sat down\( by the
side of the cooL fountain;  and when she
was bored she took a \)goLden baLL, and
threw it up on high and caught it;
and this baLL was her favorite pLaything.
```

Following are the final result and parts of the log file produced by pass 1.

in oLden times, when wishing stiLL heLped one, there Lived a king whose daughters were aLL beautifuL; and th'e youngest was so beautifuL that the sun itseLf, which has seen so much, was astonished whenever it shone in her face. cLose by the kings castLe Lay a great dark forest, and under an oLd Lime tree

in the forest' was a weLL, and when the day was very warm, the kings chiLd went out into the forest and sat down' by the side of the cooL fountain; and when she was bored she took a 'goLden baLL, and threw it up on high and caught it; and this baLL was her favorite pLaything.

Figure 5

```
Textures 1.5 (preloaded format=plain 92.6.1)
    21 OCT 1992 17:54
(test (source)
> \box255=
\vbox(158.99377+0.0)x216.81, glue set 3.04933fill
.\glue(\topskip) 3.05556
.\hbox(6.94444+1.94444)x216.81, glue set 0.45114
..\hbox(0.0+0.0)x20.0
..\tenrm i
..\tenrm n
..\glue 3.33333 plus 1.66666 minus 1.11111
..\tenrm o
..\tenrm L
..\tenrm d
......
......
..\tenrm a
..\tenrm n
..\tenrm d
..\glue 3.33333 plus 1.66666 minus 1.11111
```

```
..\tenrm t
..\tenrm h
..\hbox(0.0+0.0)x0.00002, glue set - 0.59999fil,
    shifted -6.0
...\glue 0.0 plus 1.0fil minus 1.0fil
...\hbox(0.0+0.0)x0.0
....\tensy 7
...\kern 1.2
...\glue 0.0 plus 1.0fil minus 1.0fil
..\tenrm e
........
........
..\tenrm g
..\tenrm .
..\penalty 10000
..\glue(\parfillskip) 0.0 plus 1.0fil
..\glue(\rightskip) 0.0
.\glue 0.0 plus 1.0fill

! OK (see the transcript file).
<output> {\showbox 255
           \shipout \box 255 \advancepageno }
\break ->\penalty -\@M

l.12 \vfill\eject

?
```

A summary and a wish

The methods described here have limitations and disadvantages, so they cannot be used in every situation. Method 2 still has a few unsolved problems. As a result, the macros described here cannot be canned and used 'as is'. They should be carefully studied and understood, so that they could be applied to practical problems. This means that they are beyond the grasp of beginners but, because of their power, they may provide the necessary incentive to many beginners to become full fledged wizards.

It would be so much easier to solve the three problems discussed here if the `\lastbox` command could recognize characters of text, or if a new command, `\lastchar`, were available for this purpose. This is a private wish that I hope will be shared by readers.

Finally, I would like to thank the many people who have responded to the original OTR articles of 1990. I would like to think that I was able to help some of them, and I know that their comments, questions, and criticism have helped me become more proficient in this fascinating field of OTR techniques.

References

1. Salomon, D., *Output Routines: Examples and Techniques. Part II*, *TUGboat* 11(2), pp. 212–236, June 1990.
2. Haralambous, Y., Private communication.
3. Graham, R. L., et al., *Concrete Mathematics*, Addison-Wesley, 1989.
4. Bell, E. T., *Men of Mathematics*, Simon and Schuster, 1937.
5. Knuth, D. E., *Computers and Typesetting*, vol. E, Addison-Wesley, 1986.

◇ David Salomon
California State University, Northridge
Computer Science Department
Northridge, CA 91330-8281
`dxs@secs.csun.edu`

Verbatim Copying and Listing

David Salomon

A general note: Square brackets are used throughout this article to refer to *The TEXbook*. Thus [39] refers to page 39. Also, the logo OTR stands for 'output routine', and MVL, for 'Main Vertical List'.

Introduction

Methods are developed, and macros listed, to solve the following two problems. Verbatim copying is the problem of writing a token string verbatim on a file, then executing it. Verbatim listing involves typesetting a token string verbatim, in either horizontal or vertical mode.

We start with a short review of `\edef`. In '`\edef\abc{\xyz \kern1em}`', the control sequence `\xyz` is expanded immediately (at the time `\abc` is defined), but the `\kern` command is only executed later (when `\abc` is expanded).

The same thing happens when `\abc` is defined by means of `\def`, and is then written on a file. Thus '`\write\aux{\abc}`' writes the replacement text that would have been created by `\edef\abc{...}`.

Sometimes it is desirable to write the name of a control sequence on a file, rather than its expansion. This can be done either by '`\write\aux{\noexpand\abc}`' or, similarly, by '`\write\aux{\string\abc}`'. The former form