Tutorials

Output Routines: Examples and Techniques. Part II: OTR Techniques.

David Salomon

The warnings and disclaimers in Part I* of this article also apply to this part. The methods and macros described here are not canned. They should not be copied and used verbatim. Rather, they should be carefully studied and adapted to specific needs.

The following techniques are discussed in this article, and are applied to practical situations:

1. Breaking up `\box255` in the OTR into individual lines by means of the `\last`*xx* commands.
2. Identifying individual lines or paragraphs to the OTR by means of `\rightskip`, `\parshape`, or the depth of `\box255`.
3. Attaching very small amounts of `\kern` to certain lines of text, to identify those lines to the OTR as special.
4. Placing large negative penalties at certain points in the document. This has the effect of invoking the OTR at those points. The OTR does not have to shipout anything.
5. Attaching very small vboxes below certain lines, to identify them to the OTR as special lines that require special treatment.
6. Using marks. This is a common OTR technique.
7. Setting `\vsize` to a very small value. `\box255` consists, in such a case, of just one line of text, which is then easy to examine.
8. Using a 2-pass technique where, in the first pass, certain information is written on a file, to be read by the second pass. Certain complex problems may even call for a multi-pass job.

We also remind the reader of the notation used in Part I: [...] alone makes reference to an item or items in *The TeXbook* (e.g., [400] refers to page 400 and [Ch. 6] refers to Chapter 6 in *The TeXbook*), whereas [§...] refers to a module or modules in TeX: *The Program.*

* *TUGboat* 11, no. 1, pp. 69–85.

Technique: Special Penalties

Penalties are used in TeX to control line breaks and page breaks, depending on the current mode. Penalties generated in h-mode are used by the paragraph break algorithm [§831, §859]. To communicate with the OTR by penalties, they therefore have to be generated in v-mode. A penalty of 10000 or more is considered infinite and prevents a page break. Similarly, a penalty of −10000 or less always causes a break. The idea is to say `\penalty-10001` at any point that requires the OTR's attention (TeX must be in v-mode at that time), in order to invoke the OTR at this point. A macro such as

```
\def\immed{\vadjust{\penalty-10001}}
```

can be used for this purpose. The OTR should check `\outputpenalty` and, if it equals −10001, do something special. It can then shipout `\box255` or return it to the current page.

This is a good method for communicating with the OTR, and has only one feature that makes it less than ideal; the special penalty value of −10001 does not invoke the OTR *immediately*. Instead, it is initially placed in the recent contributions, together with the rest of the paragraph, and has to wait until the page builder is exercised. The problem is that, when the page builder is exercised and the OTR invoked, TeX has already read text past the special penalty.

In a test such as

```
..\dimen0=1pt...\immed...\dimen0=2pt..\par
```

the OTR would find `\dimen0` to have a value of `2pt`.

Exercise: Write an OTR that displays the value of `\dimen0`, and perform the test above.

The reason for this behaviour is the way `\vadjust` is executed. TeX first breaks the entire paragraph into lines that are placed in the recent contributions. Only then does it place the `\vadjust` material at the proper point between two lines [259]. As a result, the OTR is invoked too late.

To solve this problem, a way should be found to exercise the page builder immediately. The page builder is exercised (see [117]) at the start and end of a paragraph; so, if the user wants to invoke the OTR at the end of a paragraph, a `\penalty-10001` is the ideal technique. The page builder is also exercised before and after a display formula, which suggests a way to exercise it inside a paragraph. The user should place, in the paragraph, a `\penalty-10001`, preceded by an empty display formula, at the point where the OTR should be invoked.

An empty formula is easy to create by `$$ $$`. Furthermore, the large flexible glues surrounding a display are easily eliminated by:

```
\abovedisplayskip=1sp
\belowdisplayskip=1sp
\abovedisplayshortskip=1sp
\belowdisplayshortskip=1sp
```

To eliminate any extra interline spaces around the display, an `\openup-\baselineskip` is placed in it. Finally, setting `\postdisplaypenalty=-10001` places the special penalty right below the display formula, to make sure that the OTR is invoked.

The result is made into a new definition of macro `\immed`:

```
\def\immed{$$\postdisplaypenalty=-10001
 \openup-\baselineskip$$}
```

The expansion `\immed` terminates the current line (same as `\hfil\break`), places an empty, invisible display formula following the line, and *immediately* invokes the OTR with `\outputpenalty` = −10001. The paragraph is not terminated.

To see the point where the formula is placed, `\immed` can be temporarily changed to:

```
\def\immed{$$\postdisplaypenalty=-10001
 \openup-\baselineskip+$$}
```

In a test such as

```
..\dimen0=1pt...\immed...\dimen0=2pt..\par
```

the OTR would find `\dimen0` to have a value of `1pt`.

This method is, again, not ideal, since it terminates the current line.

The `\last`*xx* Commands

The OTR can examine the contents of `\box255` and also break it up into its components, by means of the `\last`*xx* commands [§424, §996]. There are 4 of them: `\lastbox`, `\lastskip`, `\lastkern` and `\lastpenalty` [271]. To use those commands, the OTR should first open `\box255`, by means of an `\unvbox`. If the last item in `\box255` is a glue, its value will be reflected in `\lastskip`. Two things can be done at this point (1) `\skip0=\lastskip`; (2) `\unskip`. The first saves the glue value for future use, and the second removes it [280]. Similarly for `\lastkern` and `\lastpenalty`. If the last item is a box, the command `\setbox0=\lastbox` will both set `\box0` *and* remove the last box.

Technique: Breaking Up a Page

The OTR may use the `\last`*xx* commands in a loop, to identify successive components of `\box255`. In such a loop it is, of course, important to check at each iteration and find out what the next item is, before copying and removing it. If the next item is not a glue, `\lastskip` will have a value of `0pt`. Similarly, `\lastkern` will be `0pt`, `\lastpenalty` will be `0`, and `\lastbox` will be void. A macro `\breakup` can thus be defined, consisting of a `\loop...\repeat` to remove successive elements off `\box255`.

```
\newif\ifAnyleft \newcount\pen
\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt\else \Anylefttrue
     \skip0=\lastskip \unskip \fi
    \ifdim\lastkern=0pt\else \Anylefttrue
     \dimen0=\lastkern \unkern \fi
    \ifnum\lastpenalty=0 \else\Anylefttrue
     \pen=\lastpenalty \unpenalty \fi
    \setbox0=\lastbox
    \ifvoid0 \else \Anylefttrue \fi
  \ifAnyleft \repeat}
```

Note the use of variable `\Anyleft` to check if there is anything left in the box after each repetition of the loop. The loop repeats until none of the four items is found. The OTR simply says `\unvcopy255 \breakup`.

An alternative definition of `\breakup`, using nested `\if`s, is:

```
\newif\ifAnyleft
\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
       \setbox0=\lastbox \ifvoid0      % end of breakup loop
    \else \Anylefttrue \fi             % box encountered
    \else \Anylefttrue \unpenalty \fi  % penalty encountered
    \else \Anylefttrue \unkern \fi     % kern encountered
    \else \Anylefttrue \unskip \fi     % glue encountered
  \ifAnyleft \repeat}
```

Before discussing specific applications of the breakup technique, let's look at its main problems.

1. We have to test `\lastskip` for `0pt`. Unfortunately, TeX does not have an `\ifskip` or `\ifglue` tests. We thus have to use `\ifdim`, which tests a dimension, not a glue. The test `\ifdim\lastskip...` first converts the glue to a dimension. The problem is that such a conversion discards the stretch and shrink components of the glue [118]. Thus if the next glue item has the form `0pt plus.. minus..`, our macro will consider it zero.

The solution: change the values of certain common vertical glues that have this form to `1sp plus... minus...`. We thus declare:

```
\parskip=1sp plus1pt
\def\vfil{\vskip1sp plus1fil}
\def\vfill{\vskip1sp plus1fill}
\abovedisplayshortskip=1sp plus3pt
```

2. A similar problem exists with penalties. A math display formula is followed by a `\postdisplaypenalty` [189], whose default value is zero. As a result, any construct using the math display mode, such as `\verbatim` or `$$\vbox{\halign{...}}$$`, suffers from the same problem. The solution is to set `\postdisplaypenalty=1`.

There is also an `\interlinepenalty` parameter, which goes between the lines of a paragraph. It is usually zero but can be changed to a large value [406] to discourage a page break inside a paragraph. We set it to 1.

The above definitions are all consolidated into a new macro `\zeroToSp`, which should be used in conjunction with any page breakup.

```
\def\zeroToSp{\parskip=1sp plus1pt
  \def\vfil{\vskip1sp plus1fil}
  \def\vfill{\vskip1sp plus1fill}
  \abovedisplayshortskip=1sp plus3pt
  \postdisplaypenalty=1
  \interlinepenalty=1}
```

3. When breaking up a box using the `\last`*xx* commands, it is easy to identify the 4 types: box, glue, kern and penalty. There seems no way, however, to identify the other three components of vertical lists, namely *rules*, *marks* and *whatsits*. When our breakup loop gets to one of them, it stops, assuming that this is the end of `\box255`. A whatsit (a `\special` or a `\write`) can usually be specified in horizontal mode, which will bury it inside an `\hbox` and out of harm. A mark, on the other hand, tends to migrate outside horizontal lists [400] and into the top level of `\box255`. It therefore causes an incomplete breakup, and its use should be avoided when this technique is employed.

A similar problem is presented by a rule. An `\hrule` at the top level of a `\vbox` is considered a box [110]. However, the `\lastbox` operation cannot identify it as such, which results in an incomplete breakup.

A solution: Place the `\hrule` in its own `\vbox`, so it does not appear at the top level of the larger `\vbox`.

Partial relief: Such a case, where the breakup stops prematurely, can be detected by setting a new box (`\brk`) to the remains of `\box255` after the breakup. When the breakup stops, `\ht\brk` should be zero. An OTR can thus be written which breaks up a copy of `\box255`, and checks to see if anything is left.

```
\newbox\brk
\output={
  \setbox\brk=\vbox{\unvcopy255 \breakup}
  \ifdim\ht\brk>0pt
    \message{Incomplete breakup}\fi
  \shipout\box255 \advancepageno}
```

Exercise: Implement the above OTR and use it to typeset several pages, some containing rules or marks.

Here are a few simple applications of the breakup technique.

Duplicating a Page

Macro `\breakup` can be modified to place broken up components from `\box255` in `\box1` in the *original order*, creating a copy of the current page.

```
\zeroToSp
\newif\ifAnyleft \newcount\pen
\def\duplicate{%
 \loop \Anyleftfalse
  \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
     \global\setbox0=\lastbox \ifvoid0 % end of breakup loop
  \else \Anylefttrue                   % box present
    \global\setbox1=\vbox{\box0 \unvbox1}\fi
  \else \Anylefttrue                   % penalty present
    \pen=\lastpenalty
    \global\setbox1=\vbox{\penalty\pen\unvbox1}\unpenalty\fi
  \else \Anylefttrue                   % kern present
    \dimen0=\lastkern
    \global\setbox1=\vbox{\kern\dimen0 \unvbox1}\unkern\fi
  \else \Anylefttrue                   % skip present
    \skip0=\lastskip
    \global\setbox1=\vbox{\vskip\skip0 \unvbox1}\unskip\fi
 \ifAnyleft \repeat}
```

A test such as:

```
\newbox\brk
\output={
  \setbox\brk=\vbox{\unvcopy255 \duplicate}
  \ifdim\ht\brk>0pt
    \message{Incomplete breakup}\fi
  \shipout\box255 \shipout\box1
  \advancepageno}
```

is particularly interesting. It typesets pairs of pages, with the same page numbers. Two *physical pages* are printed for each *logical page* generated. The two pages in a pair are duplicates of each other, but are they identical?

It turns out that they are not. The main difference between `\box255` and `\box1` is their heights. The heights are different because of the flexible glues on the page. Normally, `\box255` contains some flexible vertical glues. Those glues are flexed to adjust `\ht255` to equal `\vsize`. When `\box255` is opened, however, *the glues return to their natural size.*

This can easily be seen by a test such as:

```
\newbox\brk
\output={
  \setbox\brk=\vbox{\unvcopy255 \duplicate}
  \ifdim\ht\brk>0pt
    \message{Incomplete breakup}\fi
  \message{[\the\ht255:\the\ht1]}
  \shipout\box255 \shipout\box1
  \advancepageno}
```

```
\parskip=6pt plus6pt minus6pt
\input source
\bye
```

The `\parskip` glue is given a lot of flexibility, and the heights are shown in the log file. Such a test also shows that the last pair of pages may differ a lot in their heights. This is because the last page of a document is normally only partly full, and has a `\vfill` glue at the bottom. When `\box255` is opened, the `\vfill` returns to its natural size, which is `0pt`.

How can we make sure that the two pages in a pair have the same heights? The simplest approach is to flex `\box1` in the OTR, just before it is shipped out, by saying `\setbox1=\vbox to\vsize{\unvbox1}`. Now the two pages in a pair have exactly the same height and the same glue set ratio; they are identical. Our OTR thus becomes:

```
\newbox\brk
\output={
  \setbox\brk=\vbox{\unvcopy255 \breakup}
  \ifdim\ht\brk>0pt
    \message{Incomplete breakup}\fi
  \setbox1=\vbox to\vsize{\unvbox1}
  \shipout\box255 \shipout\box1
  \advancepageno}
```

Two `\showbox` commands can temporarily be placed in the OTR to dump `\box1` and `\box255` onto the log file, and verify that they have identical components. It is important to (temporarily) increase the value of `\showboxbreadth`. Also, to make the dumps more manageable, `\vsize` should be set to a small value, such as `1in`.

Reversing a Page

It is now trivial to modify the definition of `\duplicate`, so that it breaks up items from `\box255` and places them in `\box1` *in reverse order*. This is, perhaps, a useless operation but, since our aim is to gain an understanding of OTRs, let's ask ourselves how `\box255` and `\box1` differ.

1. They are the reverse of each other, which means that each glob of `\baselineskip` glue which used to be below a line of text, is now above it. The interline spacing in `\box1` is thus all wrong. This is not very noticeable when the entire page is typeset with the same font. Mixing different font sizes, however, results in a funny looking reversed page. Also, the `\parskip` glues are misplaced but, since they are normally zero, this is not noticeable. Changing `\parskip` to some non-zero value results in large spaces following the first line of each paragraph (which are last lines on the reversed page).

2. They have different vertical dimensions. The height of `\box255` is `\vsize` and its depth is usually the depth of the last line of text. `\box1`, on the other hand, ends with `\topskip`, which is glue and thus has no depth, so `\dp1=0pt`. Also, `\box1` starts with the bottom line of `\box255`. To guarantee that `\ht1+\dp1` equals `\ht255+\dp255`, we should force `\ht1` to be the sum `\ht255`+`\dp255`.

Exercise: Write a macro `\reversepage` to reverse `\box255` into `\box1`.

Counting the Lines

The `\breakup` macro can now easily be modified to count the number of lines of text in `\box255`. We assume that `\box255` does not contain rules, marks or whatsits, and we break it up, counting the number of `\hbox`es found. Items that we don't want to count should be placed in a `\vbox`. The macros are:

```
\zeroToSp
\newif\ifAnyleft \newcount\lineCount
\def\countlines{\global\lineCount=0
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \setbox0=\lastbox \ifvoid0
    \else \Anylefttrue \ifhbox0 \global\advance\lineCount by1 \fi \fi
    \else \Anylefttrue \unpenalty \fi
    \else \Anylefttrue \unkern \fi
    \else \Anylefttrue \unskip \fi
  \ifAnyleft \repeat}

\newbox\brk
```

```
\output={\setbox\brk=\vbox{\unvcopy255 \countlines}
         \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
         \message{\the\lineCount}
         \shipout\box255 \advancepageno }
```

Breaking Up a Line of Text

Can we use the same technique to break up individual lines of `\box255`? It seems easy to define a macro `\hbreakup` that would use `\last`*xx* commands to break up a line of text. Unfortunately, this does not work, because a line of text contains individual characters, which the `\lastbox` command cannot recognize as boxes. It is interesting to note that a character of text is, in general, considered a box [63] but, evidently, there are differences between a general box and a character box. One such difference is that a character box cannot appear in a vertical list [110]. Another difference is the one mentioned above, concerning `\lastbox`, and this difference is easy to verify with a test such as

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

which shows `\box1` to be void, and typesets 'ABC'. In contrast, the test:

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

shows `\box1` to consist of an hbox with the 'C', and typesets only 'AB'.

This is an unfortunate situation. The ability to break up a line of text would have meant a full and complete communication with the OTR. The user could hide, e.g., a strut with a special depth in the line, and the OTR could easily find it, and do something with, or add something to, the line at that point. The strut could even have been left in the line.

Technique: Using `\rightskip`

Even though `\lastbox` cannot be used to break up a line of text, `\lastskip` can be used to detect glue at the right end of such a line. This suggests a way to identify certain lines to the OTR. How can a glob of glue be placed at the end of a line? It turns out that TEX places the `\rightskip` glue at the end of every line of text when the paragraph is broken into lines. The `plain` format value of `\rightskip` is `0pt` so, setting `\rightskip=1sp` will not be visually noticeable and can be used to communicate with the OTR. Unfortunately "TEX uses the same `\rightskip` value in all lines of a paragraph" [393]. This method can thus be used to identify certain paragraphs, but not individual lines, to the OTR.

An application demonstrating this technique is shown later. It has to do with 'special boxes' in a textbook. Following are two examples that are not developed in detail, since they are easier to do in other ways:

1. Suppose that a vertical rule should be typeset on the left margin of certain paragraphs. The OTR can do this by placing a rule, the size of a strut, on the left of each line that ends with `\rightskip=1sp`. However, this is easier to do by typesetting the paragraph in a `\vbox` and placing a rule on the left of the box.

2. If only one or two lines of the paragraph appear on (the bottom of) the page, we want to move them to the next page, and to `\vfill` up the present one. This can be done by the OTR checking the rightskip glue of the bottom line or two. However, it may be easier to do with `\filbreak` [111].

Technique: Using `\parshape`

If we want the OTR to do something special with, say, the second line of a paragraph, we can identify this line by making it `1sp` longer or shorter than the other lines. This can easily be done with `\parshape`. Again, there are no practical applications as yet for this technique.

Technique: Using the Depth of `\box255`

The following quote, from [400], is relevant to this technique: "Perhaps the dirtiest trick of all is to communicate with the OTR via the *depth* of `\box255`." After mastering the techniques described here, the reader will agree that this is no longer the dirtiest trick, but is a special case of the breakup technique. Examples of applications of this technique are:

1. In certain old religious texts, if a chapter ends on a certain page, and less than half a page remains, the next chapter should start on the following page; otherwise, it should start on the same page.

2. Business contracts usually consist of clauses. In certain legal situations it is desirable to break a page between clauses. If the page must be broken inside a clause, a special footer should be typeset, saying Continued.... This can be done by ending each clause with `\endclause`, a macro defined as `{\unskip\vrule height0pt width0pt depth3.5002pt}`. The `\unskip` backspaces over any possible space preceding the special strut, thus making sure that the strut will end up on the same line as the preceding word.

The OTR simply tests

```
\ifdim\dp255=3.5002pt \else
  \footline={\hfil\sevenrm Continued...}
\fi
```

3. Certain lines should not appear at the bottom of the page. A business contract is again a good example. If a certain line contains the most important words or money sums in the contract, it should better not appear at the bottom of the page, where it is less visible.† Again, a special strut can be used to identify the line and, if the OTR detects such a line, it should alert the user, who can then correct the situation by rewording the document, or by moving things around.

Technique: Communications by Kerns

Small amounts (a few sp worth) of `\kern` can be placed between the lines of text, and detected by the OTR when breaking up `\box255`. The problem is that a kern is discardable, so we have to make sure that our special kern is not discarded. The general rule is that a page can be broken at a kern only if the kern is immediately followed by glue. We, therefore, will have our special kern followed by another kern. In fact, we will place two consecutive, identical pieces of special, small kern after the text line that we want to identify to the OTR. This is done by `\vadjust{\kern1sp\kern1sp}`, which places the kerns immediately below the current line, i.e., they are placed between the line and the `\baselineskip` that normally follows it. If the line should be followed by a penalty, the order is: the line of text, the pair of kerns, the penalty, the `\baselineskip`. The places where a page can be broken have been mentioned earlier.

† Beware! Certain businessmen do just this.

Practical Examples of OTRs

The techniques described earlier, plus a few others, are now applied to practical problems.

Example: Start a Chapter On a New Page

The problem*: If a chapter ends on a certain page, and less than half a page remains, skip the rest of the page; otherwise, start the new chapter on the same page.

Solution: Macro `\chapter` is expanded at the start of each chapter. It appends a special line to the end of the preceding chapter (only if there is a preceding chapter), and invokes the OTR by an `\eject`. The special line consists of just a small `\hbox` with a rule of depth `1sp`, and width and height zero.

Each time the OTR is invoked, it checks to see if `\dp255=1sp` and `\ht255<0.5\vsize`. If yes, the OTR returns `\box255` to the MVL (it is an end-of-chapter and more than half a page remains); otherwise, `\box255` is shipped out (either less than half a page remains or not end-of-chapter).

Actually, the details are a bit more involved. If `\dp255=1sp`, then `\box255` contains text, followed by a `\vfill`, and by the special box. The last two items have to be removed before `\ht255` can be tested. To do this —

1. `\box 255` is opened, the special box at the bottom is removed by a `\lastbox` (see later), and the `\vfill` is skipped over by an `\unskip`. The result then goes back in `\box255`. The new `\box255` now has just the original text, and its height can be measured.

2. If `\ht255<0.5\vsize`, `\box255` is opened, and a message (`unv`) goes in the log file. Otherwise, a new box is shipped, consisting of `\box255`, a `\vfill`, and a footline. The size of the new box is `\vsize+12pt`, and it has to be explicitly specified.

A listing of the macros follows. They are kept, as usual, simple.

* Proposed by Robert Batzinger.

```
\newif\ifFirstCh \FirstChtrue \newif\ifRet \newcount\chnum

\newdimen\Hvsize \Hvsize=\vsize \divide\Hvsize 2
\newdimen\Nvsize \Nvsize=\vsize \advance\Nvsize 12pt
```

```
\def\chapter#1\par{\advance\chnum 1
                   \ifFirstCh \FirstChfalse
                   \else \vfill\nointerlineskip
                         \hbox{\vrule width0in height0pt depth1sp}
                         \eject \fi
                   \bigskip\noindent{\bf\the\chnum.\ #1}
                   \medskip}

\footline={...}

% if \dp255 = 1sp: unvbox255, lastbox (the line with dp = 1sp),
%    skip over the \vfill by \unskip, and return to MVL.

\output={\Retfalse
         \ifdim\dp255=1sp
           \setbox255=\vbox{\unvbox255 \setbox0=\lastbox \unskip}
           \ifdim\ht255<\Hvsize \Rettrue \fi \fi
         \ifRet \unvbox255 \message{unv}
         \else
           \shipout\vbox to\Nvsize{\box255\vfill\line{\the\footline}}
           \advancepageno \message{ship} \fi
         }
```

Example: A Religious Hymn

One way of communicating with the OTR, proposed in [App. D], is by the use of special penalty values. Any penalty value ≤ -10000 will cause the OTR to be invoked. Values < -10000 can therefore be used to tell the OTR to do something special.

Note that the OTR is not invoked when TeX first sees the penalty. It is only invoked when the page builder detects the penalty, while moving items from the recent contributions to the current page [§1005].

The OTR should check the value of variable `\outputpenalty`. If it is < -10000, it should do something special and then return `\box255` to the MVL without shipping out anything (a dead cycle). If, however, `\outputpenalty` equals -10000, the OTR should do a normal `\shipout`.

The example shown here* has to do with typesetting a religious hymn. A hymn consists of one chorus and a number of stanzas. The chorus is usually printed after the first stanza and is sung after each stanza. The problem is that a long hymn may occupy more than one page and, in such a case, the chorus should be printed on the top of each successive page.

* Proposed and solved by Robert Batzinger.

The solution is to write macros that will typeset a copy of the chorus if we are still within the same hymn, but have moved to a new page. The original text of the chorus is saved in a `\toks` variable, so it can be used as often as necessary.

Macro `\hymn` is expanded at the start of each hymn. It invokes the OTR with penalty -10001 and the OTR, in that case, simply saves the current page number in the count variable `\oldpage`. Note that the OTR does not shipout anything, and returns `\box255` to the MVL.

Macro `\stanza` is expanded at the start of each stanza. It invokes the OTR with penalty -10002. The OTR then tests `\ifnum\pageno>\oldpage` (we have moved a page or two since the last printing of the chorus) and sets the boolean variable `\prtCorus` to true. The OTR then returns `\box255` to the MVL. Macro `\stanza` tests `\prtCorus` and, if it is true, invokes macro `\setchorus` to typeset the chorus.

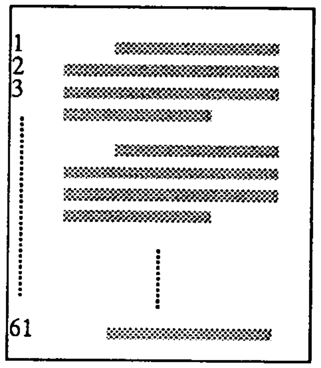
Here are the macros used:

```
\newif\ifprtCorus \newcount\oldpage
\hsize=3.5in \vsize=2.2in

\def\hymn#1#2{\bigbreak\bigskip
              \noindent{\bf #1. #2}\nobreak\medskip
```

```
               \nobreak \penalty-10001}

\def\stanza#1\endstanza{\medbreak
                        \vbox{\noindent#1}\medskip\penalty-10002
                        \ifprtCorus \setchorus\fi }

\def\chorus#1\endchorus{\toks2={#1}\setchorus}

\def\setchorus{\medskip
               \moveright.5in\vbox{\noindent
                         \hbox to 0pt{\hss\bf Chorus:\ }%
                         \the\toks2\medskip}
               \global\prtCorusfalse }

\output={\ifnum\outputpenalty=-10001
            \global\oldpage=\pageno
            \global\prtCorusfalse
            \unvbox255
           \else \ifnum\outputpenalty=-10002
            \ifnum\pageno>\oldpage
              \global\prtCorustrue \global\oldpage=\pageno \fi
            \unvbox255
           \else
             \shipout\vbox{\box255\smallskip \line{\the\footline}}
             \advancepageno
           \fi \fi }
```

Note that this is just a demonstration of a principle. The macros presented here are simple and will not always work. One case where they fail is when a hymn starts at the end of a page, and the chorus is typeset on the following page. The chorus will, in such a case, be typeset twice on that page. There may be other problems, but the idea in this article is to keep the macros simple and easy to read.

Exercise: Generalize the above macros so that they do not typeset the chorus on an odd-numbered (right hand) page if it was typeset on the preceding even-numbered (left hand) page. This way the chorus would be typeset only once on a pair of facing pages.



Line Numbers on the Margin
Figure 1

Example: Line Numbering

When writing a draft of a book, a thesis, or a report that should be reviewed by someone else, it is useful to number the lines on each page (see Figure 1). This way the reviewer can easily refer to, e.g., *line 48, page 84*. The numbers should be placed in the margin, so they can be suppressed in the final version without any changes in the layout of the document.

The method used here counts the number of lines of text by counting the boxes that make up the page. Macro `\countlines` below assumes that each box on the page is a line of text and should be numbered. Alternatively, if certain items on the page should not be numbered, they can be placed in vboxes, and `\countlines` revised to count only hboxes.

```
\zeroToSp
\newif\ifAnyleft \newcount\lineCount
\def\countlines{%
  \global\lineCount=0
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \setbox0=\lastbox \ifvoid0
    \else \Anylefttrue \global\advance\lineCount by 1 \fi
    \else \Anylefttrue \unpenalty \fi
    \else \Anylefttrue \unkern \fi
    \else \Anylefttrue \unskip \fi
  \ifAnyleft \repeat}
```

Note that, in an \halign, each line becomes an \hbox and is, therefore, counted separately. Also note that a blank line preceding a display equation becomes an empty paragraph, and is therefore counted.

The OTR breaks up a copy of the page, removing the lines of text one by one. At the same time, a new box, \box1, is built, from the bottom up, with the line numbers on the margin. For each line removed from the page, its height and depth are measured, and a line with the same size, containing the appropriate number, is added to the top of \box1. Each glue or kern removed from the bottom of the page is added to the top of \box1. At the end, the height and depth of \box1 are set to zero and it is typeset, superimposed on the original page.

```
\newcount\pen
\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0 % end of breakup loop
    \else \Anylefttrue \appendline \fi
    \else \Anylefttrue \pen=\lastpenalty
          \global\setbox1=\vbox{\penalty\pen \unvbox1} \unpenalty \fi
    \else \Anylefttrue \dimen0=\lastkern
          \global\setbox1=\vbox{\kern\dimen0 \unvbox1} \unkern \fi
    \else \Anylefttrue \skip0=\lastskip
          \global\setbox1=\vbox{\vskip\skip0 \unvbox1} \unskip \fi
  \ifAnyleft \repeat}

\def\appendline{%
  \setbox2=\hbox{\vrule height\ht0 depth\dp0 width0pt \sevenrm\the\lineCount}
  \global\advance\lineCount-1
  \global\setbox1=\vbox{\box2 \unvbox1}}

\newbox\brk
\output={\global\lineCount=0
         \setbox\brk=\vbox{\unvcopy255 \countlines}
         \global\setbox1=\vbox{}
         \setbox\brk=\vbox{\unvcopy255 \breakup}
         \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
         \ht1=0pt \dp1=0pt
         \shipout\vbox{\moveleft20pt\box1 \box255}
         \advancepageno}
```

This example illustrates both the power of the breakup technique, and its main problem. The problem is the flexible glues in `\box255`. They are flexed, by the page builder [§668, §1017], to adjust `\ht255` to `\vsize`. However, when `\box255` is opened, for the breakup, the flexible glues return to their natural size.

A partial solution is to reduce, or even eliminate, the flexibility of those glues (mainly `\parskip`). This, however, handicaps the page builder in its most important task, namely, finding a good point to break a page.

Exercise: Implement an alternative approach to the line numbering problem. The new approach should build, in `\box1`, a duplicate of `\box255` with the line numbers inserted on the left.

Example: Special Footnote Numbering

Another practical problem* is to number the footnotes in a document by the line number on the page. This problem is solved here several times, using different approaches. Each approach illustrates different OTR techniques, and also involves certain difficulties.

* Proposed by Lothar Meyer-Lerbs.

The following quote, from the Chicago Manual of Style (see also [125]), is relevant. "Since it is impossible to foresee how [footnotes] will happen to come out in the make-up, it is impracticable to number them from 1 up on each page. The best way is to number them consecutively throughout an article or by chapters in a book." The problem tackled here is much more complicated than the one proposed in the quote, and demonstrates the power of OTRs in TeX.

A Simple, Wrong Approach

The first approach is simple and intuitive. Macro `\Nfootnote` uses a penalty of −10001 to invoke the OTR prematurely. The macro is expanded from h-mode, and it has to place the penalty at the top level of `\box255`, between lines of text. This is done with `\vadjust`. The OTR breaks up `\copy255` and counts the number of lines in the page so far. It then returns `\box255` to the MVL. Macro `\Nfootnote` again takes over and typesets the footnote with the number calculated by the OTR.

The macros are very simple:

```
\def\Nfootnote#1{%
  \vadjust{\penalty-10001}%
  \footnote{$^{\the\lineCount}$}{#1}}

\zeroToSp
\newbox\brk \newif\ifAnyleft \newcount\lineCount

\def\breakup{%
  \global\lineCount=0
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0
    \else \Anylefttrue \ifhbox0 \global\advance\lineCount1 \fi \fi
    \else \Anylefttrue \unpenalty \fi
    \else \Anylefttrue \unkern \fi
    \else \Anylefttrue \unskip \fi
  \ifAnyleft \repeat}

\output={\ifnum\outputpenalty=-10001
           \setbox\brk=\vbox{\unvcopy255 \breakup}
           \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
           \unvbox255
         \else \plainoutput \fi
         }
```

but they don't work! The serious reader should, by now, know the reason. The `\vadjust` with the special penalty does not invoke the OTR *immediately*. Instead, the penalty is placed following the current line. Thus, in the second part of `\Nfootnote`, when it expands `\footnote`, the OTR has not yet been invoked.

A 2-pass Method

The idea in the second approach is to modify the OTR so that it writes `\the\lineCount` on a file. This leads to a 2-pass job, shown below.

```
\newcount\lineCount \newbox\brk \newbox\sav \newcount\pass

\newread\aux \immediate\openin\aux=\jobname.lin
\ifeof\aux \immediate\openout\aux=\jobname.lin \pass=1 \message{pass 1}
\else \pass=2 \message{pass 2} \fi

\newif\ifAnyleft
\zeroToSp

\def\countlines{%
  \global\lineCount=0
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0
    \else \Anylefttrue \global\advance\lineCount by1 \fi
    \else \Anylefttrue \unpenalty \fi
    \else \Anylefttrue \unkern \fi
    \else \Anylefttrue \unskip \fi
  \ifAnyleft \repeat}

\output={\ifnum\outputpenalty=-10001
           \ifnum\pass=1
             \setbox\brk=\vbox{\unvcopy255 \countlines}
             \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
             \immediate\write\aux{\the\lineCount} \fi
           \unvbox255 % return to MVL
         \else \plainoutput \fi
         }
% shipout with footnotes

\def\Nfootnote#1{%
  \ifnum\pass=1 \vadjust{\penalty-10001}\footnote*{#1}%
  \else \read\aux to\tmp \footnote{$^{\tmp}$}{#1}\fi}
```

In the first pass, macro `\Nfootnote` creates the special penalty and also expands `\footnote*{...}` to typeset the footnote, so it takes the right amount of space on the page. In the second pass, the macro reads the correct number off the file, and invokes `\footnote` with that number. This is still simple and usually works.

It may fail, however, in cases where a footnote appears close to the bottom of the page. Imagine a footnote on line 60 of page 4. Because of the `\penalty-10001` following this line, TeX will invoke the OTR with a 60-line page. The OTR will (1) write the line count, 60, on the file; (2) return `\box255` (with the 60 lines) to the current page, removing the special penalty. Since the current page is now large, TeX immediately starts looking for a good page break. It may decide, since the special penalty isn't there any more, to break the page after line 59. Line 60 thus becomes line 1 of the next page, but the number 60 has already been written on the file.

Another 2-pass Solution

The third approach is similar except that, instead of being written on a file, the line numbers are saved—by the OTR—in memory. This makes sense since there usually aren't many footnotes on any single page. In the second pass, macro `\Nfootnote` uses this information to expand `\footnote` with the correct line numbers. This approach suffers from the same problem as the previous one, but it is shown here because it illustrates how to save the line numbers, each as an `\hbox`, in a large `\vbox`. Extracting them later is easily done with a `\lastbox`.

Note that the 2-pass structure is different from the previous one. Previously, each pass was a separate TEX job, and the line numbers were saved on a file between the jobs. In the present method, however, the line numbers are saved in a box (`\sav`), which is stored in memory and thus disappears at the end of the job. The two passes must, therefore, be done in the same job. This is faster but requires the source text to be `\input` from a separate file.

```
\newcount\lineCount \newbox\brk \newbox\sav \newif\ifAnyleft
\zeroToSp

\def\breakup{%
  \global\lineCount=0
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0
    \else \Anylefttrue \ifhbox0 \global\advance\lineCount1 \fi \fi
    \else \Anylefttrue \unpenalty \fi
    \else \Anylefttrue \unkern \fi
    \else \Anylefttrue \unskip \fi
  \ifAnyleft \repeat}

% pass 1
\output={\ifnum\outputpenalty=-10001
           \setbox\brk=\vbox{\unvcopy255 \breakup}
           \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
           \global\setbox\sav=\vbox{\hbox{\sevenrm\the\lineCount}\unvbox\sav}
           \unvbox255
         \else \plainoutput \fi
         }
% The above line should later be changed to:
% \setbox0=\box255 \deadcycles=0,
% since we don't really want to shipout pages in pass 1.

\def\Nfootnote#1{%
  \vadjust{\penalty-10001}%
  \footnote*{#1}}

\input source \vfill\eject \pageno=1

% pass 2
\output={\ifnum\outputpenalty=-10001
           \unvbox255
         \else \plainoutput \fi
         }

\def\Nfootnote#1{%
  \setbox\sav=\vbox{\unvbox\sav \global\setbox0=\lastbox}%
```

```
   \footnote{\raise4pt\copy0}{#1}}

  \input source
```

A Complex, 3-pass Approach

Approach 4: A three-pass job. The first pass determines the line numbers (throughout the document) of lines with footnotes. Those numbers are saved in a `\vbox` called `\Asav`. The second pass counts the number of lines on each page. Those numbers are also saved, in another box, `\Bsav`. The third pass uses the numbers from the two boxes to determine the correct line numbers and to typeset the footnotes. This is complex and, perhaps, can be done in a simpler way. Nevertheless, it has the advantage of demonstrating several useful OTR techniques.

Before describing the 3 passes in detail, here is a simple numeric example: Let's assume that we have three pages, with 50, 30 and 40 lines respectively. There are footnotes on lines 3, 15, 15 and 44 of the first page, and lines, 25 and 34 of the third page. Pass 1 will save the numbers 3, 15, 15, 44, 105 and 114 in `\Asav` (note that 15 occurs twice). In pass 2, the line counts 50, 30 and 40, of the 3 pages are saved in `\Bsav`. Pass 3 starts by extracting the 50 from `\Bsav`. The first 4 times macro `\Nfootnote` is expanded, it extracts the numbers 3, 15, 15 and 44 from `\Asav`. Those numbers are ≤ 50, so they are used for numbering the first 4 footnotes. The fifth expansion extracts 105 from `\Asav`. This is > 50, so the next number, 30, is extracted from `\Bsav` and added to the 50. The current footnote number, 105, is still > 80, so the next number, 40, is extracted from `\Bsav` and added to the 80. The current footnote number, 105, is now ≤ 120, so 80 is subtracted and the result, 25, is used. The last number is 114, again ≤ 120, so again 80 is subtracted, yielding 34.

The steps in each pass are:

Pass 1. Macro `\Nfootnote` computes a running number for each footnote, and creates a `\mark` with that number. The footnote itself is not typeset, but `\Nfootnote` typesets an asterisk to occupy space on the line, approximately equal to that taken by the final footnote number. `\vsize` is set to a small value, so the OTR receives a `\box255` with just one line [400], which makes it easy to number the lines throughout the document. Each time the OTR is invoked, it checks `\firstmark`, `\botmark` and compares them to `\topmark`. This way it knows if there are any footnotes on the line. If there are any, the line number is saved in box `\Asav`, once for each footnote on the line.

```
\newcount\temp \newcount\footno \newcount\lineno \newbox\Asav

\def\Nfootnote#1{\advance\footno 1 \mark{\the\footno}*} % typeset an *

\output={\global\advance\lineno 1
         \temp=\botmark
         \advance\temp -\firstmark
         \advance\temp 1
         \ifnum\firstmark\botmark \ifnum\topmark\firstmark \temp=0 \fi \fi
% \temp is now the number of footnotes on the current page (one line)
         \ifnum\temp>0
          \loop
            \global\setbox\Asav=\vbox{\vskip\lineno sp \null\unvbox\Asav}
            \advance\temp-1
          \ifnum\temp>0 \repeat
         \fi
         \setbox0=\box255 % get rid of
         \deadcycles=0
         }

%      *** Executable commands ***
\message{Pass 1;} \vsize=10pt % small value
```

```
23. \footno=0 \lineno=0 \setbox\Asav=\vbox{}
24. \input source \eject
```

Lines 1–19 are macro definitions, and declarations of variables. Lines 22–24 are the actual commands executed in pass 1.

\vsize is set, on line 22, to the small value 10pt. The page in \box255 will, as a result, consist of just one line of text.

The OTR calculates \temp, on lines 6–8, as \botmark − \firstmark + 1. \temp is now the number of footnotes on the current page (which consists of just one line of text). However, if \botmark = \firstmark = \topmark, there are no footnotes on the current line, and \temp is set, on line 9, to 0.

If \temp $\neq 0$, the loop, on lines 12–15, saves variable \lineno on top of \box\Asav as glue (in units of sp).

Pass 2. Macro \Nfootnote typesets each footnote with an asterisk. No marks are used. \vsize is set to its normal value, and the OTR breaks up a copy of each page, counts the number of lines, and saves that number, as the top glue item, in box \Bsav.

```
 1. \newif\ifAnyleft \newbox\Bsav \newbox\brk
 2.
 3. \def\Nfootnote#1{\footnote*{#1}}
 4. % typeset the footnote so it occupies the right space on the page
 5.
 6. \def\countlines{%
 7.   \global\lineno=0
 8.   \loop \Anyleftfalse
 9.     \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
10.       \global\setbox0=\lastbox \ifvoid0
11.     \else \Anylefttrue
12.           \ifhbox0 \global\advance\lineno1 \fi \fi % count \hboxes on the page
13.     \else \Anylefttrue \unpenalty \fi
14.     \else \Anylefttrue \unkern \fi
15.     \else \Anylefttrue \unskip \fi
16.   \ifAnyleft \repeat}
17.
18. \output={\setbox\brk=\vbox{\unvcopy255 \countlines}
19.          \ifdim\ht\brk>0pt \message{Incomplete breakup}
20.                            \showboxbreadth=1000 \showbox\brk \fi
21.          \global\setbox\Bsav=\vbox{\vskip\lineno sp \null\unvbox\Bsav}
22.          \plainoutput
23.          }
24.
25. % *** Executable commands ***
26. \zeroToSp
27. \message{Pass 2;} \setbox\Bsav=\vbox{}
28. \vsize=2in %  or any desired value
29. \input source \vfill\eject \pageno=1
```

This is a simple pass. It is again divided into declarations and macro definitions (on lines 1–23), and executable commands (on lines 27–29).

Note the \plainoutput on line 22. This causes pages to be shipped out in pass 2, in addition to the final pages created by pass 3. We end up with two sets of pages that should be identical, except for the footnote numbers. Because of the problem mentioned later, the pages may not be identical, and it is therefore important to compare the two sets before they are printed. When the results are finally printed, the pages created by pass 2 should, of course, be suppressed.

Pass 3. Count variable `\lineshiped` is set to zero. Count variable `\totalines` is set to the first value of `\Bsav` (50 in our example). `\vsize` remains at its normal value. The OTR ships out pages in the normal way. Each time `\Nfootnote` is invoked it (1) extracts the next item from `\Asav` into `\lineno`; (2) if `\lineno` ≤ `\totalines`, the footnote is created with `\lineno` − `\lineshiped`; (3) otherwise, `\lineshiped` is set to `\totalines` and the next number is extracted from `\Bsav` and added to `\totalines`. Step (2) is repeated.

```
1. \newcount\totalines \newcount\lineshiped
2.
3. \def\compare{%
4.   \ifnum\lineno>\totalines
5.     \global\lineshiped=\totalines
6.     \global\setbox\Bsav=
7.       \vbox{\unvbox\Bsav \setbox0=\lastbox \global\temp=\lastskip \unskip}%
8.     \global\advance\totalines by \temp
9.   \expandafter\compare % expand recursively for each page w/o footnotes
10.  \fi}
11.
12. \def\Nfootnote#1{%
13.   \setbox\Asav=
14.     \vbox{\unvbox\Asav \setbox0=\lastbox \global\lineno=\lastskip \unskip}%
15.     % extract bottom glue into \lineno
16.   \compare
17.   \advance\lineno -\lineshiped
18.   \footnote{$^\the\lineno$}{#1}}
19.
20. \output={\plainoutput}
21.
22. %  *** Executable commands ***
23. \message{Pass 3;} \lineshiped=0
24. \setbox\Bsav=
25.   \vbox{\unvbox\Bsav \setbox0=\lastbox  \global\totalines=\lastskip \unskip}
26. \input source
27. \bye
```

Macro `\compare`, lines 3–10, expands itself recursively to implement the (pseudo-code) loop

```
while \lineno>\totalines
  \lineshiped:=\totalines
  extract \temp from \Bsav
  \totalines:=\totalines+\temp
end while;
```

The `\expandafter` on line 9 makes sure that the `\fi`, on line 10, is gobbled up by TEX before `\compare` is recursively expanded. Without the `\expandafter`, the `\fi` would be saved in a stack and popped out at the end of the recursion. In case of a deep recursion, that could overflow the stack.

The macros are deliberately kept simple and readable and, as a result, are not completely general, and don't work in all cases. One such case is where there are no footnotes on the first page; there may be other cases. However, in general, this *approach* seems to work, and seems to have just one, small problem. Passes 1 and 2 typeset an asterisk '$*$', in the body of the text, where each footnote should be. This is done to occupy space on the line, space that, in pass 3, is taken by the footnote number. Passes 1 and 2 thus end up with the same line breaks *but pass 3 may not.* The problem is that footnote numbers, in our case, are one or two digits, and thus may be slightly wider or narrower than the '$*$'. This may, in rare cases, cause different line breaks in pass 3, leading to wrong footnote numbers.

Exercise: Why is it true that footnote numbers, in our case, can be one or two digits, but not three?

Saving Numbers in a vbox. An interesting point is that our line numbers are saved as *glue* in a `\vbox`. This is done by `\vskip\lineno sp \null`. The `sp` is necessary since, otherwise, the value of

`\lineno` would be converted to scaled points. The `\null` is an empty `\hbox` to separate the individual pieces of glue in the large `\vbox`. This technique can only be used if the total number of footnotes in the document is not too large. For a large number of footnotes, there may not be enough room in memory for our boxes, and a file should be used (in our case, two files).

The actual saving of the count variable `\lineno` in box `\Asav` is done by:

```
\global\setbox\Asav=
  \vbox{\vskip\lineno sp \null\unvbox\Asav}
```

Extracting the bottom glue item from `\Asav` is done by:

```
\setbox\Asav=
  \vbox{\unvbox\Asav \setbox0=\lastbox
        \global\lineno=\lastskip \unskip}
```

Example: Tables Broken Across Pages

Another practical problem*: In a document with a lot of tables, many times a table is split over two pages. In such a case, the OTR should typeset "Continued..." at the bottom of the page.

Two approaches are shown, one using marks and the other, special boxes, to communicate with the OTR.

The first approach: A `\mark{Continued...}` is inserted at the start of each `\halign` (following the preamble), and a `\mark{}` is inserted just before the end of the table.

The output routine simply typesets `\botmark` at the bottom of the page, using the right font. The following macros are used:

```
\output={\shipout\vbox{\box255
  \smallskip\line{\sevenrm\hfil\botmark}
  \smallskip\line{\the\footline}}
  \advancepageno}
\def\beginCont{\mark{Continued...}}
\def\endCont{\mark{}}
```

and a typical table looks like:

```
\halign{...preamble...\cr \beginCont
...1st line...\cr
...
...last line...\endCont\cr}
```

Note that the first mark becomes part of the first table entry (column 1 row 1). The last mark, similarly, becomes part of the last table entry (last column bottom row). This means that, sometimes, the mark may be locked inside an internal box. For instance, if the preamble says `$#$`, then the mark will be buried in the math box. Generally this creates no problem but, if the mark is buried too deeply in `\box255`, it may not be discovered [259] during `\shipout`.

* Proposed by Mary McClure.

A partial remedy is to use `\noalign{\beginCont}` or `\noalign{\endCont}`, depending on which mark is missing during `\shipout`. This way, the mark precedes (or follows) the entire table. The table, in such a case, should end up with ...⟨*last line*⟩`...\cr\endCont}`. These constructs should be used only in an emergency, since they also may fail. A typical example is a table that starts at the top of a page. Its `\mark{Continued...}` may, in such a case, be the last thing in the preceding page.

An interesting feature of this method is the even page height. Each page shipped out contains a line with the `\botmark`, and this line occupies the same amount of space on the page, regardless of the size of the mark. Thus if the line preceding the mark has a depth of `1.94444pt`, and the mark contains the text Continued..., (which has a height of `4.78334pt`), the `\baselineskip` glue is set at `5.27222pt`. This separates the baselines by $1.94444 + 5.27222 + 4.78334 = 12$pt. However, if the mark is empty, and the line preceding it has a depth of `0.8333pt`, the `\baselineskip` glue right above the mark is set at `11.1667pt`, again separating the baselines by $0.8333 + 11.1667 + 0 = 12$pt.

Communication by Special `\vboxes`

The second approach uses a `\vbox` with a special depth to communicate with the OTR. This looks promising, especially since the `\vboxes` on both sides of a table can be attached to it by means of a `\nobreak` (which is essentially a `\penalty10000`). The implementation is similar to the preceding case.

```
\def\beginCont{\noalign{\vbox{
 \hrule width0pt height0pt depth1sp}
 \nobreak}}
\def\endCont{\noalign{\nobreak\vbox{
 \hrule width0pt height0pt depth2sp}}}
```

Note that the `\nobreak` in `\beginCont` *follows* the special `\vbox`, while that in `\endCont` *precedes* it.

```
\zeroToSp
\newif\ifAnyleft

\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \setbox0=\lastbox \ifvoid0
    \else \Anylefttrue
        \ifvbox0
          \ifdim\dp0=1sp \Anyleftfalse \global\toks0={Continued...}
          \else\ifdim\dp0=2sp \Anyleftfalse \global\toks0={}\fi \fi
        \fi \fi
    \else\Anylefttrue \unpenalty \fi
    \else \Anylefttrue \unkern \fi
    \else \Anylefttrue \unskip \fi
  \ifAnyleft \repeat}

\newbox\brk
\output={\setbox\brk=\vbox{\unvcopy255 \breakup}
         \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
         \shipout\vbox{\box255\smallskip
                       \line{\sevenrm\hfil\the\toks0}
                       \smallskip\line{\the\footline}}
         \advancepageno
         }
```

This works! Note, however, that macro `\breakup` stops when it finds the first special `\vbox`. In such a case, there is no point in finishing the break up of `\box255`. This method therefore generates many "Incomplete breakup" messages, and the user should make sure that the text and the tables should not contain any of the things that normally stop the breakup.

Exercise: A variation of the same problem. Each table is preceded by a header. If the table is broken across pages, the header should be typeset at the top of the second page.

Exercise: Add a parameter to macro `\beginCont` above. The macro should now create a `\vbox` whose depth is the value of the parameter, in scaled points. Modify macro `\breakup` such that it will save different messages in `\toks0` depending on the depth of the special boxes found.

Example: Verse Numbers in the Left Margin

The problem*: In the Bible, each chapter is divided into verses. If a verse starts on a certain line, we want the verse number typeset on the left margin of the line. Also, if two or more verses start on the same line, a range of verse numbers, such as 23–24 should be typeset on the left margin.

* Proposed by Robert Batzinger.

Solution: Each verse starts with an expansion of macro `\verse`. The macro computes the verse number and typesets it in the body of the text. In addition, it uses a `\vadjust` to generate a special `\vbox` and to attach it, with a `\penalty10000`, right below the line of text in `\box255`. The special box has a height and width of zero, and a depth equal to the verse number in scaled points. A line of text can thus be followed by any number of such boxes, and no page break can occur in that area. The verse numbers are stored in the `\count` variables `\fVerse` (final verse) and `\sVerse` (start verse).

The OTR expands macro `\breakup`, which breaks up `\box255` and transfers its components to `\box1`. On identifying a special `\vbox`, macro `\breakup` expands `\verseline` which (1) converts the depth of the special box into a count; (2) checks for another special box and converts its depth into another count; (3) removes the line of text above the special boxes, attaches the verse number(s) (via `\Label`) as an `\llap`, and adds the result to `\box1`.

After the breakup is complete, the OTR ships out `\box1`.

```
\newcount\sVerse \newcount\fVerse \newif\iftwo
\def\verseline{%
  \fVerse=\dp0 \unpenalty
  \global\setbox0=\lastbox
  \ifvoid0 \message{error1;}\fi
  \twofalse
  \ifvbox0 \ifdim\dp0<500sp \ifdim\dp0>0sp \twotrue \fi \fi \fi
  \iftwo
    \sVerse=\dp0 \unpenalty
    \def\Label{\hbox to.4in{\hfil\the\sVerse--\the\fVerse\hfil}}
    \global\setbox0=\lastbox
    \ifvoid0 \message{error2;}\fi
  \else\def\Label{\hbox to.4in{\hfil\the\fVerse\hfil}}\fi
  \global\setbox1=\vbox{\line{\llap{\sevenrm\Label\kern6pt}\box0}\unvbox1}
  }

\newif\ifAnyleft \newcount\pen \newif\ifverseBox

\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0 % end of breakup loop
    \else \Anylefttrue \verseBoxfalse
      \ifvbox0 \ifdim\dp0<500sp \ifdim\dp0>0sp \verseline \verseBoxtrue \fi\fi\fi
      \ifverseBox \else \global\setbox1=\vbox{\box0\unvbox1}\fi \fi
    \else \Anylefttrue \pen=\lastpenalty
          \global\setbox1=\vbox{\penalty\pen\unvbox1} \unpenalty \fi
    \else \Anylefttrue \dimen0=\lastkern
          \global\setbox1=\vbox{\kern\dimen0\unvbox1} \unkern \fi
    \else \Anylefttrue \skip0=\lastskip
          \global\setbox1=\vbox{\vskip\skip0\unvbox1} \unskip \fi
  \ifAnyleft \repeat}

\newbox\brk
\output={\setbox\brk=\vbox{\unvbox255 \breakup}
         \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
         \setbox1=\vbox to\vsize{\unvbox1}
         \shipout\box1 \advancepageno}

\newcount\versno \versno=0
\def\verse{%
  \advance\versno by 1
  \hskip1em{\bf\the\versno: }\nobreak
  \vadjust{\nobreak\vbox{\hrule width0pt height0pt depth\the\versno sp}}}

\zeroToSp
\input source
\bye
```

Problems With This Approach

1. To keep our macros simple, they are limited to at most two verses per line. However, it is easy to generalize `\verseline` to handle up to 3 verses. It is also possible, although probably not necessary, to generalize it to handle any number of verses per line.

Exercise: Disregarding the statement above, generalize `\verseline` to handle any number of verses per line. This requires recursive calls to identify and remove any number of consecutive special `\vbox`es below a text line.

2. The verse numbers are typeset on the left margin, centered in an `\hbox to .4in`. This is wide enough for 3-digit verse numbers. For larger numbers, it may be necessary to enlarge that box. If no centering is required, then it is enough to say `\def\Label{\the\sVerse--\the\fVerse}`.

3. The verse numbers always start from 1. It is possible to let the user specify a start number by:

```
\message{enter start verse number:}
\read16to\ent
\versno=\ent
```

instead of `\versno=0`.

4. The macros recognize a special box if its depth is positive and is less than 500`sp`. In case of many verses, the 500 should be changed to a larger value. The following quote (from [400]) is reassuring: "A distance of 1000sp is invisible to the naked eye."

Example: Verse Numbers, An Alternative Method

Here is an alternative method that does not transfer components from `\box255` to `\box1`. It breaks up a copy of `\box255` and, each time a special box (or several consecutive special boxes) is discovered, the macro measures the height of the remaining copy, and uses the height to build, in `\box1`, the range of verse numbers in the margin. When the breakup is completed, `\box1` looks like a skeleton with just the verse number ranges. The OTR then superimposes the two boxes by: `\shipout\hbox{\llap{\box1}\box255}`. (A similar method is used on [391–392].)

Here are the steps in detail:

```
\newcount\versno \versno=0
\def\verse{\advance\versno by 1
  \hskip1em{\bf\the\versno: }\nobreak
  \vadjust{\nobreak\vbox{
    \hrule width0pt height0pt
           depth\the\versno sp}}}
```

Macro `\verse` creates a special `\vbox` whose depth equals the verse number (in scaled points), and attaches it, with a `\nobreak`, below the line where the verse starts.

The output routine copies `\box255` into `\box2` and expands `\Obreak` to break up `\box2` and create the necessary information in `\box1`. It then invokes `\reversebox` to break up `\box1` and build, again in `\box2`, the correct skeleton. Both steps are described below. The final step is to shipout a superposition of `\box2` and `\box255`.

```
\output={\setbox2=\copy255 \Obreak
  \ifdim\ht\brk>0pt
    \message{Incomplete breakup}\fi
  \reversebox \setbox2=
    \vbox to\vsize{\unvbox2\vfil}
  \wd2=0pt
  \shipout\hbox{\llap{\box2}\box255}
  \advancepageno}
```

Macro `\Obreak` expands `\breakup` to break up `\box2` and, if another verse (another special box) is found, the height of the remaining `\box2` is placed, as `\kern`, in `\box1`, and `\Obreak` expands itself recursively. The process repeats until no more verses are found on the page.

```
\newbox\brk
\def\Obreak{%
  \setbox\brk=\vbox{\unvbox2 \breakup}
  \ifanotherverse
    \global\anotherversefalse
    \global\setbox1=
      \vbox{\unvbox1\kern\ht\brk}
    \setbox2=\box\brk
    \expandafter\Obreak
  \fi}
```

Macro `\breakup` loops and breaks up items from `\box2` until it reaches the end, or until it finds an item that is a `\vbox` with a depth in the range 0–500`sp`. If it finds such an item, it expands `\verseline`.

```
\newif\ifAnyleft
\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0 % end of breakup loop
    \else \Anylefttrue
      \ifvbox0 \ifdim\dp0<500sp\ifdim\dp0>0sp \verseline \Anyleftfalse \fi\fi\fi
      \fi
    \else \Anylefttrue \unpenalty \fi
    \else \Anylefttrue \unkern \fi
    \else \Anylefttrue \unskip \fi
  \ifAnyleft \repeat}
```

Macro `\verseline` removes the `\penalty10000` that precedes the special box, and checks to see if there is another special box right above it. If there is one, the box and the penalty above it are removed, and the boolean variable `\iftwo` is set to true.

Next, macro `\Label` is defined, as an `\hbox to.4in{\hfil` *one or two verse numbers* `\hfil}` and is inserted, as an `\llap`, into the margin of `\box1`.

```
\newcount\sVerse \newcount\fVerse \newif\iftwo \newif\ifanotherverse

\def\verseline{%
  \fVerse=\dp0 \unpenalty
  \global\setbox0=\lastbox
  \ifvoid0 \message{error1;}\fi
  \twofalse
  \ifvbox0 \ifdim\dp0<500sp \ifdim\dp0>0sp \twotrue \fi \fi \fi
  \iftwo
    \sVerse=\dp0 \unpenalty
    \def\Label{\hbox to.4in{\hfil\the\sVerse--\the\fVerse\hfil}}
    \global\setbox0=\lastbox
    \ifvoid0 \message{error2;}\fi
  \else \def\Label{\hbox to.4in{\hfil\the\fVerse\hfil}} \fi
  \global\setbox1=\vbox{\unvbox1 \line{\llap{\sevenrm\Label\kern6pt}\hfil}}
  \global\anotherversetrue}
```

At the end of the process, the output routine expands `\reversebox` to break up items from `\box1`, process them and place them in `\box2` in the correct order. To understand this process, let's imagine a page with three verses at a distance of `2in`, `3in` and `6in` from the top (Figure 2). The breakup process starts at the bottom of the page, measures the height `A` of verse 3, then `B` and, finally, `C`, creating `\box1` as in Figure 3.

However, we want a box that looks like Figures 4–5, where the `\kern`s are measured from one verse to the next, not always from the top. We also have to make sure that the lines of text do not take any vertical space, so we add a negative `\kern` after each line, to skip back to the top of the line.

```
\def\reversebox{\setbox2=\vbox{}
\ifvoid1
\else
 \dimen1=0pt \unvbox1
 \loop
  \dimen0=\lastkern \unkern
  \dimen2=\dimen0
  \advance\dimen0 by-\dimen1
  \dimen1=\dimen2
  \setbox0=\lastbox
  \dimen2=\ht0 \advance\dimen2 by\dp0
  \global\setbox2=\vbox{\unvbox2
   \kern\dimen0 \box0 \kern-\dimen2}
 \ifdim\lastkern>0pt\repeat
\fi}
```

Macro `\reversebox` contains a loop that breaks up `\box1`, calculates the quantities `C`, `B-C`, `A-B`, and places them in `\box2` with the lines of text, each followed by a negative `\kern`. When finished, The OTR appends a `\vfil` to end up with a height of `\vsize`. This way, `\box2` has the same height as `\box255`, and they can be superimposed and shipped out together.

To run the whole thing, just say:

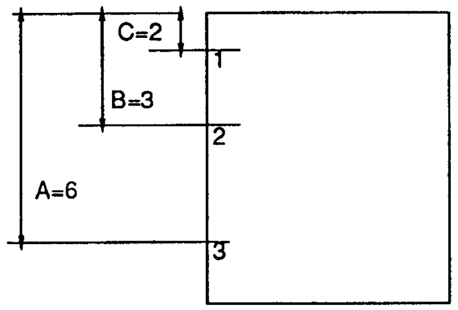


Figure 2

```
\kern 2in (=C)
\hbox{\hfil 3 \hfil}
\kern 3in (=B)
\hbox{\hfil 2 \hfil}
\kern 6in (=A)
\hbox{\hfil 1 \hfil}
```

Figure 3

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

This is, perhaps, not the most elegant solution, nor is it compact. Each macro, however, has its own, well defined, task, making it easier to read and understand the whole thing.

A 'Special Box' OTR

The problem: In many modern science texts, the main flow of text is interrupted by 'special boxes'. They can be used to develop certain topics in detail, to present a historical background of other topics, or to present the author's opinion or reminiscences. To distinguish such a box from the rest of the text, it may be surrounded by rules on all sides.

The intuitive approach is to place the special text in a `\vbox` and build the rules as in [Ex. 21.3]. This, of course, won't work since the 'special box' may have to straddle two pages, but a `\vbox` is indivisible.

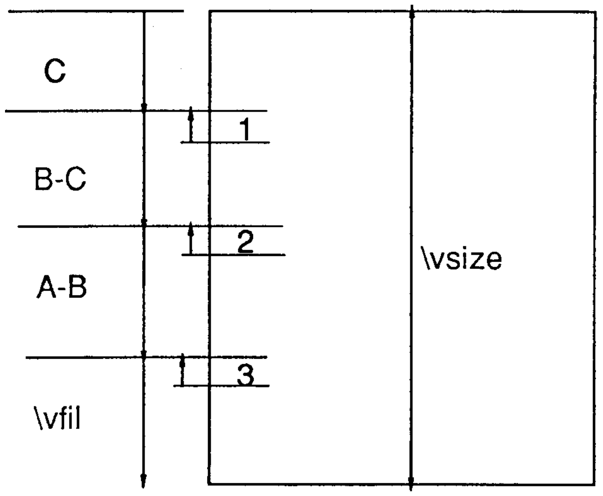


Figure 4

```
\kern 2in (=C)
\hbox{\hfil 1 \hfil}
\kern-(size of preceding line)
\kern 1in (=B-C)
\hbox{\hfil 2 \hfil}
\kern-(size of preceding line)
\kern 3in (=A-B)
\hbox{\hfil 3 \hfil}
\kern-(size of preceding line)
\vfil
```

Figure 5

The approach used here identifies the start and end of the special text by making its lines narrower. Macro `\startspbox` draws the top `\hrule` of the special box and expands `\narrower`. Macro `\endspbox` terminates the effect of `\narrower`, and draws the bottom `\hrule`. Note that the hrules are placed in boxes, since otherwise they would cause an incomplete breakup.

The OTR breaks up the page and creates a duplicate. Each narrow line (a line for which `\rightskip` > 0) is surrounded with two short rules. To make the rules on successive lines touch, the normal interline glue is suppressed when a narrow line is found.

```
\def\Hrule{\line{\vrule width\hsize height.4pt}}
\def\startspbox{\medskip \Hrule \nobreak \smallskip \begingroup \narrower}
\def\endspbox{\smallskip \nobreak \endgroup \Hrule \medskip}

\zeroToSp
```

```
\newif\ifsurround
\def\Strut{\vrule height8.5pt depth3.5pt}
\def\checkline{%
  \setbox2=\hbox{\unhcopy0
                 \ifdim\lastskip>0pt \global\surroundtrue
                 \else\global\surroundfalse\fi}}

\newif\ifAnyleft \newcount\pen
\def\specialbox{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0 % end of breakup loop
    \else \Anylefttrue
      \ifhbox0\checkline \ifsurround\setbox0=\hbox{\Strut\box0\Strut}\fi \fi
      \global\setbox1=\vbox{\box0 \unvbox1} \fi
    \else \Anylefttrue \pen=\lastpenalty
          \global\setbox1=\vbox{\penalty\pen\unvbox1} \unpenalty \fi
    \else \Anylefttrue \dimen0=\lastkern
          \global\setbox1=\vbox{\kern\dimen0 \unvbox1} \unkern \fi
    \else \Anylefttrue \skip0=\lastskip
          \ifsurround\skip0=0pt \fi % suppress the normal interline glue
          \global\setbox1=\vbox{\vskip\skip0 \unvbox1} \unskip \fi
  \ifAnyleft \repeat}

\newbox\brk
\output={\setbox\brk=\vbox{\unvbox255 \specialbox}
         \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
         \shipout\box1 \advancepageno}
```

Example: Revision Bars

Certain documents — such as the bylaws of an organization, or the user's manual for a computer system — may go through many revisions. Sometimes it is desirable to emphasize (or flag) the revised parts by placing a vertical bar on the left margin of revised lines. If the revision is short, affecting only one line, there is no need for a special OTR and a `\vadjust` like the one below, can be used (see also [Ex. 14.28]).

```
\def\rev{\vadjust{\moveleft6pt\vbox to0pt{
\kern-12pt\hrule height10pt width1pt\vss}}}
```

However, if the revision may affect more than one line, the problem becomes much more complex and the OTR should be involved.

A Simple Method. We start with a relatively simple approach,* which is sketched below, but is not implemented.

1. Macro `\beginvbars` saves the page-so-far in a box `\partialpage`.

2. Macro `\endvbars` places a bar on the left of `\box255`, appends it to `\partialpage`, and returns the whole thing to the MVL, so that a good page break can be found.

The problems with this approach are:

1. The revision may start in mid-paragraph. In such a case, the first part of the paragraph goes in box `\partialpage`, and eventually has to be seamlessly glued to the rest of the paragraph. A similar case occurs when the revision ends within a paragraph.

2. When `\box255` is appended to `\partialpage`, its `\topskip` glue should be replaced by the normal interline skip.

A Better Solution. The approach shown here is different. The start and end of each revision are flagged with small, special boxes placed between the text lines. The OTR breaks up `\box255` looking for the special boxes. The distance of each special box from the top of the page is measured. The distances are then used to prepare vertical rules in a separate box (`\box3`), which is eventually typeset on the left of `\box255`.

* Due to Amy Hendrickson.

Macro `\startrev` uses `\vadjust` to place a special `\vbox` with a height of `1sp` below the line where the revision starts. Macro `\endrev` places a similar box, with a height of `2sp`, below the last line of the revised text. Note that, if the revised text is short, the two special boxes may end up being placed, one above the other, below the same line.

```
\def\startrev{\vadjust{%
 \nointerlineskip\nobreak\vbox to1sp{}}}
\def\endrev{\vadjust{%
 \nointerlineskip\nobreak\vbox to2sp{}}}
```

Macro `\Obreak` expands `\breakup` with a copy of `\box255`. The breakup loop stops when a special box, with height = `2sp` is found. `\Obreak` then measures the height of the remaining page, stores that height in `\box1` as `\kern`, stores a flag indicating that a vertical bar should end at that point, and restarts the loop. When a box with height = `1sp` is found, `\Obreak` does a similar thing, except that it places a different flag, indicating that the bar should start at that point. The flags are special hboxes with a width of either `1sp` or `2sp`.

```
\zeroToSp

\output={\global\setbox1=\vbox{}
         \setbox2=\copy255 \Obreak
         \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
         \arrangebox
         \setbox3=\vbox to\vsize{\unvbox3\vfil} \wd3=6pt
         \shipout\hbox{\llap{\box3}\box255}
         \advancepageno}

\newbox\brk
\newif\ifstartbar \startbarfalse \newif\ifendbar \endbarfalse
\def\Obreak{%
  \setbox\brk=\vbox{\unvbox2 \breakup}
  \ifstartbar
    \global\startbarfalse
    \global\setbox1=\vbox{\unvbox1\kern\ht\brk\hbox to1sp{}}
    \setbox2=\box\brk
    \expandafter\Obreak
    \fi
  \ifendbar
    \global\endbarfalse
    \global\setbox1=\vbox{\unvbox1\kern\ht\brk\hbox to2sp{}}
    \setbox2=\box\brk
    \expandafter\Obreak
    \fi
  }

\newif\ifAnyleft
\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0
    \else \Anylefttrue
      \ifvbox0
        \ifdim\ht0=1sp \global\startbartrue \Anyleftfalse \fi
        \ifdim\ht0=2sp \global\endbartrue \Anyleftfalse \fi
      \fi \fi
    \else \Anylefttrue \unpenalty \fi
    \else \Anylefttrue \unkern \fi
```

```
      \else \Anylefttrue \unskip \fi
    \ifAnyleft \repeat}
```

At the end of the breakup loop, the OTR expands macro `\arrangebox`, which reads the kerns and flags from `\box1`, and uses them to generate the actual vertical bars in `\box3`. It uses the following algorithm:

```
PrevKern:=0;
read Kern,Flag from \box1
if Flag=start
  place \kern of size Kern-PrevKern
    in \box3
  PrevKern:=Kern, PrevFlag:=Flag;
if Flag=end
  place a rule of size Kern-PrevKern
    in \box3
  PrevKern:=Kern, PrevFlag:=Flag;
if \box1 is empty and PrevFlag=start
  place a rule of size \vsize-PrevKern
    in \box3,
End;
```

And here is a listing:

```
\newif\ifcontin
\def\arrangebox{
  \setbox3=\vbox{} \dimen1=0pt
  \loop
    \ifdim\ht1>0pt
      \setbox1=\vbox{\unvbox1
        \global\setbox0=\lastbox
        \global\skip0=\lastkern \unkern}
      \contintrue \dimen0=\wd0
      \dimen2=\skip0
      \advance\dimen2 by-\dimen1
      \dimen1=\skip0
      \ifdim\dimen0=1sp
        \setbox3=\vbox{\unvbox3 \kern\dimen2}
        \fi
      \ifdim\dimen0=2sp
        \setbox3=\vbox{\unvbox3
          \hrule height\dimen2 width1pt}
        \fi
    \else
      \continfalse
      \ifdim\dimen0=1sp \dimen2=\vsize
        \advance\dimen2 by-\dimen1
        \setbox3=\vbox{\unvbox3
          \hrule height\dimen2 width1pt}
      \fi
    \fi
  \ifcontin \repeat}
```

As usual, the macros can be improved. The user may notice that the size and placement of the bars is not ideal, and can be improved. This is especially true for cases where only one line of text is revised.

Exercise: Generalize the macros so that they can typeset a revision number, in `\sevenrm`, on the left of each bar. The number should be specified by the user, as a parameter of `\startrev`.

Summary

The examples and techniques described here, even though incomplete and simplified, demonstrate how very powerful TeX is, compared to other typesetting systems.

The main concepts behind TeX namely, boxes, glue, penalties and macros, are different from those used by other systems, and are more difficult to master. At the same time, they are more powerful, and the user who is willing to invest the time and effort necessary to learn TeX, is rewarded by high quality results.

Part III will introduce insertions and their use in OTRs. There will be a general introduction to insertions, examples of OTRs with insertions, and a description of the `plain` format OTR.

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