TEX from \indent to \par

Marek Ryćko
Wydawnictwo Do
ul. Filtrowa 1
00-611 Warszawa, Poland

Bogusław Jackowski
ul. Tatrzańska 6/1
80-331 Gdańsk, Poland

Abstract

A proper answer to even apparently simple questions about TEX can only be answered with a detailed formal specification of TEX's mechanisms, which will allow us to derive the behaviour of TEX in more complex situations.

Introduction

There are some seemingly simple questions about TEX which may be difficult to answer without precise knowledge of TEX mechanisms.

In the following section we will ask three such questions, encouraging the reader to answer them without reading the explanation.

Actually, the explanation follows immediately from a detailed specification of TEX's action at the beginning and at the end of a paragraph. We believe that if such a specification of all TEX's mechanisms existed, answers to most questions concerning behaviour of TEX would be equally simple.

The pivotal sections are 'Switching from Vertical to Horizontal Mode' and 'Switching from Horizontal to Vertical Mode'. The section 'From Input Characters to Commands' contains necessary introductory material.

Questions

In all questions we assume the normal meaning of tokens of plain TEX.

Q1. What is the difference between:
(*) `\everypar{\def\indent{1}}`
`\indent 3 is a prime number.`
and
(**) `\everypar{\def\vrule{1}}`
`\vrule 3 is a prime number.`
What is typeset in both cases and why?

Q2. Assuming that TEX is in vertical mode, what is the difference between:
(*) `\parindent=0mm \indent\par`
and
(**) `\noindent\par`
What is appended to the main vertical list and why?

Q3. What is the difference between:
(*) `\par`
and
(**) `{\par}`
What is the state of TEX after executing these commands in both cases and why?

From Input Characters to Commands

Let us start with a closer look into TEX's way of processing input data. Three levels of the processing can be distinguished:

L1. Reading characters from the input file and transforming them into tokens (lexical analysis).

L2. Expanding tokens.

L3. Executing commands; at this level TEX creates internal lists (horizontal, vertical and math lists), transforms them into boxes and writes some boxes to the DVI file (using the `\shipout` command).

Knuth talks about "eyes," "mouth" and "stomach" of TEX, etc.; we prefer to speak about "levels."

Names and meanings of tokens. In order to understand what happens at the beginning and at the end of a paragraph it is essential to be aware of the difference between names and meanings of tokens.

Following Knuth, we will denote by `*\xyz` the meaning of the command `\xyz` at the beginning of the TEX job. By `xyz` we will denote a token, the name of which consists of the letters 'xyz'. Such a token is created by TEX from the sequence of letters 'xyz' preceded by a current escape character, usually backslash.

For example, the token hbox, the *name* of which consists of the letters 'hbox,' has initially the meaning *\hbox. Saying '\let\hbox=\par' a user may change the meaning of hbox to the current meaning of par, most likely to *\par. Incidentally, TEX replaces every empty input line with the token par regardless of the meaning of this token. The meaning of par may be *\par, but par may be also, for example, a macro expanding to a sequence of tokens.

Transforming input characters into tokens. From the point of view of TEX, the input file is a sequence of characters organized into lines. TEX reads such characters one by one and transforms them at level 1 into so-called tokens, according to definite rules. For example, the following sequence of 15 input characters:

\ e n s p a c e ␣ D o n . . .

is transformed into a sequence of 7 tokens:

enspace Don...

The first one is a control sequence token and the remaining are character tokens stored by TEX along with their category codes.

Each token created at this level is associated with its current meaning which can be either *a primitive meaning* (a meaning that is built into TEX) or it can be *a macro* (a meaning that can be defined by a user in terms of other meanings). Regarding the meaning we can classify all tokens as follows:

(a) with respect to expandability as *expandable* and *unexpandable*;
(b) with respect to primitivity as *primitive* and *macros*.

The expandable tokens can be primitive, like \if, \the, \noexpand, \csname, or they can be macros defined using \def or a related assignment (\edef, \gdef, \xdef).

All unexpandable tokens are primitive. This group contains, among others: tokens like \hskip, \hbox, etc.; letters and other characters; all tokens defined by the \chardef assignment; some tokens defined by \let or \futurelet.

Expanding tokens. Level 2 of TEX, i.e., the expansion level, reads tokens from the input token list and expands them. If the first token in the input token list is expandable, level 2 of TEX expands it, that is, replaces this token (possibly with some tokens following it) with another sequence of tokens.

If — after the replacement — the first token is still expandable, the expansion is repeated until the list starts from an unexpandable token. Obviously, this process may loop infinitely.

For example, the result of expansion of the first token in the input token list:

enspace Don...

is the sequence of tokens:

kern .5em␣Don...

because the first token enspace is expandable (it is a plain TEX macro) and its expansion is '\kern.5em␣'. The token kern is unexpandable, hence no further expansion takes place.

The input token list with an unexpandable token at the beginning is submitted to level 3 of TEX.

Commands. By a *command* we mean an unexpandable (primitive) token at the beginning of the input token list. If a command may or must have arguments, only the first token is a command. For example, in the input token list:

kern .5em␣Don...

the token kern is the command and the tokens '.5em␣' are arguments. They are being read as a part of the process of executing the command.

In general, a command can read arguments from an input list either demanding expansion from level 2 or not.

Level 3 of TEX — the level that executes commands — is the central level. Every time this level is about to execute the next command it "asks" level 2 to prepare the input token list such that at the beginning of the list there is a primitive (unexpandable) token. In turn, level 2 "asks" level 1 for preparing necessary tokens.

Level 3 executes the command according to its meaning, taking into account the current internal state of TEX, including the values of various parameters, and, in particular, taking into account current TEX's *mode*.

One of the results of executing commands is creation of various kinds of internal lists. The types of lists include: horizontal, vertical and math lists.

At every moment TEX is in one of the following six modes determining what type of list it is currently constructing:

(a) vertical mode (v-mode)
(b) internal vertical mode (iv-mode)
(c) horizontal mode (h-mode)
(d) restricted horizontal mode (rh-mode)
(e) math mode
(f) display math mode

At the very beginning of a job TEX is in v-mode and all the lists are empty. A list is constructed by appending new elements to it. The process of list construction can be briefly summarized as follows: mathematical lists are converted into h-lists; an h-list created in h-mode (material for a paragraph)

is converted into a v-list and appended to a current v-list; a vertical list created in v-mode is converted to boxes by a page builder; eventually, boxes to which a command \shipout is applied are written to a DVI file.

Summary of Paragraph Construction

In the process of creating a paragraph by TEX there are three distinct phases:

P1. Switching from v-mode to h-mode (opening a new h-list — see the section 'Switching from Vertical to Horizontal Mode').

P2. Creating the h-list. (We do not discuss this phase in the paper. The notion of h-list is explained in "The TEXbook," pp. 94 - 95. The systematic description of how the commands processed in h-mode influence the state of the h-list contain chapters 24 and 25 of "The TEXbook," pp. 267 - 287).

P3. Switching from h-mode to v-mode (converting the h-list into a v-list and appending this vertical list to the main v-list; this is discussed in the section 'Switching from Horizontal to Vertical Mode').

We will focus our attention on the moment of switching from v-mode or iv-mode to h-mode and back again.

For the sake of simplicity we confine ourselves to the case where display math is not used inside a paragraph.

Switching from Vertical to Horizontal Mode

In this section we describe *when* and *how* level 3 of TEX accomplishes the change of modes from v-mode or iv-mode to h-mode.

First we say "when", i.e., we list the commands that — if executed in one of v-modes — switch TEX's state to h-mode.

Then we say "how", that is, we list the actions that TEX performs during the mode change.

Switching from vertical to horizontal mode: when. Some commands will be called here *vh-switches*, because if encountered in v-mode or in iv-mode they switch TEX to h-mode. They can be classified into two groups:

(a) explicit vh-switches:
- \indent;
- \noindent;

(b) implicit vh-switches (called by Knuth horizontal commands):
- letter: any character token of category 11 (also implicit; for example, control sequence \d after executing the assignment '\let\d=A'; the assignment associates the token d with a meaning that is primitive in TEX);
- other character: any character token of category 12 (also implicit; for example, control sequence \one after executing the assignment '\let\one=1');
- *\char;
- a "chardef" token, i.e., a control sequence or an active character which has been assigned a meaning by the command \chardef (for example, control sequence \ae after the assignment '\chardef\ae="1A'; once again, the assignment associates the token ae with a meaning that is primitive in TEX);
- *\noboundary (a new primitive that appeared in TEX 3.0);
- *\unhbox, *\unhcopy (independently of the contents of the box being an argument);
- *\valign;
- *\vrule;
- *\hskip;
- *\hfil, *\hfill, *\hss, *\hfilneg (these tokens are primitive, not macros, even though the effects they cause could be achieved using *\hskip with appropriate parameters);
- *\accent;
- *\discretionary, *\-;
- *\␣ (control space *\␣ is a primitive command and if used in v-mode switches the mode to horizontal; note that normal space ␣, in general any space token, is ignored in v-mode);
- $ (also the first $ of the pair $$ starting the displayed math formula).

It should be stressed that commands *\hbox, *\vbox and *\vtop are not switches. Such commands encountered in v-mode do not change the mode. The box (preceded by proper glue) is appended to the current v-list.

Switching from vertical to horizontal mode: how. Assume that TEX is in either v-mode or iv-mode. When level 3 encounters a vh-switch at the beginning of the input token list it performs in turn the following actions:

(a) Optionally, a vertical glue \parskip is appended to the vertical list:
- if TEX is in iv-mode and the list is empty, the glue is not appended,

— if TEX is in iv-mode and the list is not empty, the glue is appended,
— if TEX is in v-mode the glue is always appended to the part called "recent contributions" of the main v-list.

(b) If TEX is in v-mode (not iv-mode) the page builder is exercised, that is TEX runs the algorithm that moves elements of the v-list from the part of "recent contributions" to the part "current page". In particular it may cause page breaking (running the `\output` routine).

(c) Switching from v-mode or iv-mode to h-mode occurs.

(d) Variables `\spacefactor` and `\prevgraf` are assigned values 1000 and 0, respectively (these assignments are called by Knuth "global intimate assignments" and work in a rather peculiar way).

(e) A new h-list is initialised in the following way:
— if the vh-switch that caused the mode change was *`\noindent`, the newly created h-list is empty;
— if the vh-switch that caused the mode change was anything else (*`\indent` or any horizontal command), an empty box of width *`\parindent` is put at the beginning of the h-list.

(f) The following elements are appended to the beginning of the input token list:
— the contents of the token register `\everypar` (normally this register is empty),
— the vh-switch, provided it is a horizontal command; thus the explicit vh-switches *`\indent` and *`\noindent` are *not* put back into the input token list.
The rest of the input token list remains unchanged.

(g) Execution of the commands from the input token list starts. The commands are supplied by level 2 of TEX.

Answer to the Question Q1

Let us recall the question **Q1** of the first section. We have asked about the difference between

```
(*) \everypar{\def\indent{1}}
    \indent 3 is a prime number.
```

and

```
(**) \everypar{\def\vrule{1}}
     \vrule 3 is a prime number.
```

From the point (f) of the list of actions performed by TEX at the beginning of a paragraph (see subsection 'Switching from vertical to horizontal mode: how') we can draw the following conclusions: if a paragraph has started from the `\indent` command, the token `indent` is not put back into the input token list, therefore after executing the actions (a)-(f) the input token lists differ in both cases.

In the case (*) the list is: '`def` `indent` `{1}3␣is␣a␣prime␣number.`'; in the case (**) the list contains one more token: '`def` `vrule` `{1}` `vrule` `3␣is␣a␣prime␣number.`'.

Since redefining `indent` has nothing to do with the remainder of the list, the typesetting result in the case (*) will be "3 is a prime number."

In the case (**) the token `vrule` is first defined as a macro expanding to the token 1 and then the newly defined macro `vrule` is expanded to 1. Therefore in this case the result will be "13 is a prime number."

This example shows some of consequences of the rule that the explicit vh-switches (`indent` and `noindent`) are not put back into the input token list after switching to h-mode.

Switching from Horizontal to Vertical Mode

When level 3 of TEX executes commands in h-mode, some commands cause closing the h-list and performing some actions that lead to switching from h-mode to v-mode.

In the following subsection we say *when* TEX switches from h-mode to v-mode, i.e., we list the commands that cause switching. Then we explain *how* this mode change is performed.

Switching from horizontal to vertical mode: when. The commands listed below are called hv-switches, because if executed in h-mode they usually cause TEX to complete the h-mode and switch back to the enclosing v-mode or iv-mode. Similarly to the case of vh-switches, there are two groups of switches:

(a) explicit hv-switches:
— *`\par` (any token the current meaning of which is the same as the meaning of the token `par` when TEX starts a job);

(b) implicit hv-switches (called by Knuth vertical commands):
— *`\unvbox`;
— *`\unvcopy`;
— *`\halign`;
— *`\hrule`;
— *`\vskip`;
— *`\vfil`;
— *`\vfill`;

— *\vss;
— *\vfilneg;
— *\end;
— *\dump.

Switching from horizontal to vertical mode: how. The behaviour of TEX when it reads a hv-switch heavily depends on the type of the switch. If the switch is a vertical command (implicit hv-switch), TEX proceeds as follows:
— it inserts a token par at the beginning of the input token list (*before* the hv-switch token), regardless of the meaning of the par token;
— it starts executing commands from the input list (possibly expanding par if currently it is a macro).

It should be emphasized that TEX *does not change* the mode before reading the token par and that the expanded meaning of par may redefine the token that triggered the action (please note the danger of looping).

If the switch is explicit (*\par), TEX "truly" finishes the paragraph, performing all or some of the actions (a)-(h) listed below.

TEX's behaviour depends on whether the h-list is empty or not at the moment. If the h-list *contains at least one element*, all of the actions (a)-(h) are performed. If the h-list *is empty*, only the actions marked with an asterisk are executed, i.e., (e), (g) and (h).

All possible actions are:
(a) discarding the final element of the h-list, provided it is glue or leaders;
(b) appending to the end of the h-list the following three elements:
  — \penalty10000 (forbid break),
  — glue of the size \parfillskip,
  — \penalty-10000 (force break);
(c) fixing the line-breaking parameters to be used in the next step,
(d) breaking h-list into lines and transforming this list into a v-list being the sequence of boxes, glue, penalty items and possibly other elements;
*(e) switching from h-mode back to the enclosing v-mode or iv-mode;
(f) appending the v-list created in step (d) to the enclosing v-list;
*(g) restoring the basic values of the parameters:
  — \parshape=0, \hangindent=0pt, \hangafter=1 (influencing the shape of a paragraph),
  — \looseness=0 (influencing the number of lines of a paragraph);
*(h) exercising the page builder if the current mode is the v-mode (but not iv-mode), i.e., initiating the process of moving elements from the recent contribution part of the vertical list to the current page.

Answer to the Question Q2

The question was:

What is the difference between:
(*) \parindent=0mm \indent\par
and
(**) \noindent\par

Recall that we start in v-mode. The assignment of (*) '\parindent=0mm' is just an assignment and does not append anything to the v-list. In both cases the command switching to h-mode (\indent or \noindent) causes appending the vertical glue of the size \parskip to the vertical list.

The command \par works differently in both cases (see subsection 'Switching from horizontal to vertical mode: how') because h-lists constructed are different:
(*) h-list at the moment of executing of the \par command contains a box of width 0mm,
(**) h-list at the moment of executing of the \par command is empty (the \noindent command does not append anything to the h-list).

So, according to what has been said in subsection 'Switching from horizontal to vertical mode: how', points (a) and (b), in the case (*) TEX 'breaks into lines' a list containing:
— the empty box,
— \penalty10000,
— \parfilskip glue,
— \penalty-10000.

The result is a one-line paragraph that is appended to the v-list as a single box preceded by a \parskip glue and an interline glue.

In the case (**) only the \parskip glue is appended to the vertical list, since the h-list is empty at the time the \par command is executed.

Answer to the Question Q3

We have asked what was the state of TEX after (*) executing \par and after (**) executing {\par}.

As we already know, TEX reacts to the command *\par performing the sequence of actions listed in subsection 'Switching from horizontal to vertical mode: how'. The results of most of the actions do not depend on the current level of grouping. However, the assignments mentioned in (g) are local within the current group.

Normally, at the end of each paragraph, TEX sets the values of \parshape, \hangindent, \hangafter and \looseness to 0, 0pt, 1 and 0 respectively. But if a paragraph ends with {\par} instead of \par these values are assigned locally within the group surrounding \par. After closing the group TEX restores the values that the parameters had before the group started.

So, if the parameters mentioned above had standard values before \par or {\par}, their values do not change in both cases. If at least one of these parameters had a nonstandard value before \par or {\par}, executing just the \par command would result in restoring the standard value of this parameter, while in the case of {\par} the value of this parameter would be the same as before.

For example, by redefining \par as {\endgraf} and separating paragraphs with blank lines one can conveniently retain the same \parshape for several consecutive paragraphs.

Conclusions

We would like to emphasize that it is not the questions and answers mentioned in this paper that are important.

Our goal was to convince the reader that having a detailed (or, even better, formal) specification of TEX's mechanisms one could easily deduce the behaviour of TEX in all situations.

We have described here a small fragment of TEX's machinery. Although the description is only partial and not fully precise, we believe that it makes a lot of mysterious reactions of TEX understandable and straightforward.

Acknowledgements

Tomek Przechlewski and Piotr Pianowski: thank you.