Letters

Truth in Indexing

Jonathan Fine reports in *TUGboat* 13 #4, page 495, under the heading "Too Many Errors", that Knuth is in error when he uses the index-reminder scheme described on pages 424–425 of *The TEXbook*. Padding the page numbers is irrelevant; check Knuth's original version in the lower display on page 424, where you will surely notice the `\noexpand` to which Fine objects. As a matter of fact, Knuth himself produced just the "error" Fine describes when preparing the index for *Concrete Mathematics*, at least in the first impression of that volume. Look up Christian Goldbach in the index for the first printing; the page number given there is 583 but Goldbach is only mentioned in the first line of page 584.

I wonder if Fine has overlooked the paragraph divided between pages 424 and 425 of *The TEXbook*, in which Knuth describes his philosophy regarding index construction. He may also have overlooked several references in the tutorials in volumes 10, 11, and 12 of *TUGboat*, to my belief that automation can sometimes be carried to excess. Indeed, the effort required to avoid the necessity for proof reading one's document by automating all of its components that might be subject to automation will often be significantly greater than the effort needed to accomplish the task with reasonable restraint in this regard. Wherever we concentrate our efforts, we are still required to pay close attention to the results generated by them.

The only printings of *Concrete Mathematics* that I have seen, so far, are the first and sixth. Studying the differences between them can be both entertaining and informative, even in connection with the bibliography and the index. Between these two printings, two additional items were interpolated before the entry that originally appeared at the top of page 584 of the bibliography, hence the problem with which Fine was so concerned vanished as a result of natural causes. There are at least two possibilities: One's book is so popular that it must be reprinted frequently and in the process trivial errors that do not vanish automatically are easily fixed; if the book doesn't require reprinting, it may be that it has few readers or none, in which case the distinction between gross errors and trivial errors simply evaporates.

Let me propose, for further discussion in this context, what I would like to call the Occam-Ludd Razor: *Entities should not be multiplied beyond necessity, and automation should be encouraged when it simplifies things and avoided when it does not.*

Lincoln Durst
46 Walnut Road
Barrington, RI 02806 U.S.A.
LKD@Math.AMS.org

Macros

Letter-Spacing in TEX

Philip Taylor

One of the joys of looking at a page of TEXset material, particularly when compared to the majority of today's magazines, is the uniform grayness of the page. Whereas many of today's top-end DTP packages frequently achieve justification through the use of letter-spacing, TEX prefers to distribute any spare white space between words rather than between letters. Indeed, there *are* no intrinsic facilities within TEX which would permit the use of letter-spacing, even were it desired.

And yet, there *are* times when letter-spacing is effective: in running heads, for example, or for mastheads or titles. In some languages, letter-spacing (then more properly termed *Sperrsatz*) is used for stress or emphasis, much as we use italicisation in English. For these purposes, then, rather than as a general letter-spacing tool, I have developed the following code, which allows at most a single line of text to be letter-spaced. It is worth pointing out straight away that there are some restrictions on the text, although considerably fewer than in earlier releases: it should not, for example, contain unprotected `\accents`, (neither explicit, using the `\accent` primitive, nor implicit, through the use of control symbols as `\'`), although either form may be used provided that the accent and its accompanying letter are concealed within a brace-delimited group; control-sequences without arguments may occur in the text to be typeset, but if they expand to text, that text will not be letter-spaced, and thus it is difficult, although not impossible, to typeset THE JOY OF LETTER-SPACED TEX! (And, of course, it should contain no lower-case text: "A man who would letter-space lower-case

text would probably steal sheep [Goudy]"). But these are the only restrictions: provided that the text is straightforward, considerable flexibility is offered in the degree of letter-spacing achieved.

The design desideratum was very simple: create a `\letterspace` macro which would provide *at least* the same degree of flexibility in determining the degree of letter-spacing as TEX already provides for the specification of `\hboxes` and `\vboxes`, and using the same syntax. Thus we must allow `\letterspace` {*text*}, `\letterspace to` *dimen* {*text*} and `\letterspace spread` *dimen* {*text*}. In addition, I sought to provide even greater flexibility, by providing the user with additional information concerning the text to be letter-spaced: in particular, its natural height, depth and width. These are all made available through reserved TEX control sequences: `\naturalheight`, `\naturaldepth` and `\naturalwidth`. By combining the two, a very elegant syntax is provided for letter-spacing text: for example, `\letterspace to 1.5 \naturalwidth` {*text*}, or `\letterspace spread 0.5 \naturalwidth` {*text*} (both of which have the same effect).

The code is presented twice: once as a monolithic entity, so that the reader can take in its structure at a glance, and once in annotated form, so that its inner workings can be clarified; first the code as monolithic entity:

```
%%% Open control sequences

\newdimen \naturalwidth
\newdimen \naturaldepth
\newdimen \naturalheight


%%% Concealed control sequences

\expandafter \chardef
        \csname \string @code\endcsname =
                \the \catcode `\@
\catcode `\@ = 11

\newbox \l@tterspacebox
\newtoks \l@tterspacetoks

\def \sp@ce { }
\def \hsss
    {\hskip 0 pt plus 1 fill minus 1 fill\relax}

\let \@nd = \empty
\let \@x = \expandafter

\let \sp@cetoken = \relax
\edef \t@mp {\let \sp@cetoken = \sp@ce}
\t@mp \let \t@mp = \undefined


%%% Primary (user-level) macro

\def \letterspace #1#%
    {\def \hb@xmodifier {#1}%
     \afterassignment \l@tterspace
     \l@tterspacetoks =
    }


%%% Secondary (implementation-level) macro

\def \l@tterspace
    {\setbox \l@tterspacebox = \hbox
                {\the \l@tterspacetoks}%
     \naturalwidth =  \wd \l@tterspacebox
     \naturaldepth =  \dp \l@tterspacebox
     \naturalheight = \ht \l@tterspacebox
     \hbox \hb@xmodifier
     \bgroup
           \hss
           \@x \l@tt@rspace
           \the \l@tterspacetoks {}\@nd
           \hss
     \egroup
    }


%%% Tertiary (implementation-level) macro

\def \l@tt@rspace #1\@nd
    {\ifx  \@nd #1\@nd
           \let \n@xt = \relax
     \else
           \p@rtition #1\@nd
           \ifx \h@ad \sp@ce \hsss \h@ad \hsss
           \else
                \h@ad
                \ifx \t@il \@nd \else \hsss \fi
           \fi
           \@x \def \@x \n@xt \@x
                {\@x \l@tt@rspace \t@il \@nd}%
     \fi
     \n@xt
    }


%%% Adjunct macros -- list partitioning

\def \p@rtition #1\@nd
    {\m@kespacexplicit #1\@nd
     \@x \p@rtiti@n \b@dy \@nd
    }

\def \p@rtiti@n #1#2\@nd
    {\def \h@ad {#1}%
     \def \t@il {#2}%
    }


%%% Adjunct macros -- <space>... -> {<space>}...
```

```
\def \m@kespacexplicit #1\@nd
    {\futurelet \h@ad \m@kesp@cexplicit #1\@nd}

\def \m@kesp@cexplicit #1\@nd
    {\ifx \h@ad \sp@cetoken
          \@x \def \@x \b@dy
                     \@x {\pr@tectspace #1\@nd}%
     \else
          \def \b@dy {#1}%
     \fi
    }%

\@x \def \@x \pr@tectspace \sp@ce #1\@nd {{ }#1}


%%% re-instate category code of commercial-at

\catcode `\@ = \the \@code
```

The same code is now presented in annotated form, each group of declarations, and each individual macro, being preceded by a discussion of their purpose and functionality:

As far as is possible, 'inaccessible' control sequences (i.e. control sequences containing one or more commercial-at s) are used to minimise the risk of accidental collision with user-defined macros; however, the control sequences used to access the natural dimensions of the text are required to be user accessible, and therefore must contain only letters.

```
%%% Open control sequences

\newdimen \naturalwidth
\newdimen \naturaldepth
\newdimen \naturalheight
```

All control sequences defined hereafter are inaccessible to the casual user, including the control sequence used to store the category code of commercial-at; this catcode must be saved in order to be able to re-instate it at end of module, as we have no idea in what context the module will be `\input`. Having saved the catcode of commercial-at, we then change it to 11 (i.e. that of a letter) in order to allow it to be used in the cs-names which follow.

```
%%% Concealed control sequences

\expandafter \chardef
        \csname \string @code\endcsname =
                \the \catcode `\@
\catcode `\@ = 11
```

We will need a box register in order to measure the natural dimensions of the text, and a token-list register in which to save the tokens to be letter-spaced.

```
\newbox \l@tterspacebox
\newtoks \l@tterspacetoks
```

We will need to test whether a particular macro expands to a space, so we will need another macro which does so expand with which to compare it; we will also need a 'more infinite' variant of `\hss`.

```
\def \sp@ce { }
\def \hsss
    {\hskip 0 pt plus 1 fill minus 1 fill\relax}
```

Many of the macros will use delimited parameter structures, typically terminated by the reserved control sequence `\@nd`; we make this a synonym for the empty macro (which expands to nothing), so that should it accidentally get expanded there will be no side effects. We also define a brief synonym for `\expandafter`, just to keep the individual lines of code reasonably short.

```
\let \@nd = \empty
\let \@x = \expandafter
```

We will also need to compare a token which has been peeked at by `\futurelet` with a space token; because of the difficulty of accessing such a space token (which would usually be absorbed by a preceding control word), we establish `\sp@cetoken` as a synonym. The code to achieve this is messy, because of the very difficulty just outlined, and so we 'waste' a control sequence `\t@mp`; we then return this to the pool of undefined tokens.

```
\let \sp@cetoken = \relax
\edef \t@mp {\let \sp@cetoken = \sp@ce}
\t@mp \let \t@mp = \undefined
```

The user-level macro `\letterspace` has exactly the same syntax as that for `\hbox` and `\vbox`, as explained in the introduction; the delimited parameter structure for this macro ensures that everything up to the open brace which delimits the beginning of the text to be letter-spaced is absorbed as parameter to the macro, and the brace-delimited text which follows is then assigned to the token-list register `\l@tterspacetoks`; `\afterassignment` is used to regain control once the assignment is complete.

```
%%% Primary (user-level) macro

\def \letterspace #1#%
    {\def \hb@xmodifier {#1}%
     \afterassignment \l@tterspace
     \l@tterspacetoks =
    }
```

Control then passes to `\l@tterspace`, which starts by setting an `\hbox` containing the text to be typeset; the dimensions of this box (and therefore of the text) are then saved in the open control sequences previously declared, and the `\hbox` becomes of no further interest.

A new `\hbox` is now created, in which the same text, but this time letter-spaced, will be set; the box starts and ends with `\hss` glue so that if only a single character is to be letter-spaced, it will be centered in the box. If two or more characters are to be letter-spaced, they will be separated by `\hsss` glue, previously declared, which by virtue of its greater degree of infinity will completely override the `\hss` glue at the beginning and end; thus the first and last characters will be set flush with the edges of the box.

The actual mechanism by which letter-spacing takes place is not yet apparent, but it will be seen that it is crucially dependent on the definition of `\l@tt@rspace`, which is expanded as the box is being set; the `\@x` (`\expandafter`) causes the actual tokens stored in `\l@tterspacetoks` to be made available as parameter to `\l@tt@rspace` rather than the token-list register itself, as it is these tokens on which `\l@tt@rspace` will operate. The `\@nd` terminates the parameter list, and the `{}` which immediately precedes it ensures that that there is always a null element at the end of the list: without this, the braces which are needed to protect an accent/character pair would be lost if such a pair formed the final element of the list, at the point where they are passed as the second (delimited) parameter to `\p@rtiti@n`; by definition, TEX removes the outermost pair of braces from both simple and delimited parameters during parameter substitution if such braces form the first and last tokens of the parameter, and thus if a brace-delimited group ever becomes the second element of a two-element list, the braces will be irrevocably lost. The `{}` ensure that such a situation can never occur.

```
%%% Secondary (implementation-level) macro

\def \l@tterspace
    {\setbox \l@tterspacebox = \hbox
                {\the \l@tterspacetoks}%
     \naturalwidth =  \wd \l@tterspacebox
     \naturaldepth =  \dp \l@tterspacebox
     \naturalheight = \ht \l@tterspacebox
     \hbox \hb@xmodifier
     \bgroup
           \hss
           \@x \l@tt@rspace
           \the \l@tterspacetoks {}\@nd
           \hss
     \egroup
    }
```

The next macro is `\l@tt@rspace`, which forms the crux of the entire operation. The text to be letter-spaced is passed as parameter to this macro, and the first step is to check whether there is, in fact, any such text; this is necessary both to cope with pathological usage (e.g. `\letterspace {}`), and to provide an exit route, as the macro uses tail-recursion to apply itself iteratively to the 'tail' of the text to be letter-spaced; when no elements remain, the macro must exit.

Once the presence of text has been ensured, the token-list representing this text is partitioned into a head (the first element), and the tail (the remainder); at each iteration, only the head is considered, although if the tail is empty (i.e. the end of the list has been reached), special action is taken.

If the first element is a space, it is treated specially by surrounding it with two globs of `\hsss` glue, to provide extra stretchability when compared to the single glob of `\hsss` glue which will separate consecutive non-space tokens; otherwise, the element itself is yielded, followed by a single glob of `\hsss` glue. This glue is suppressed if the element is the last of the list, to ensure that the last token aligns with the edge of the box.

When the first element has been dealt with, the macro uses tail recursion to apply itself to the remaining elements (i.e. to the tail); `\@x` (`\expandafter`) is again used to ensure that the tail is expanded into its component tokens before being passed as parameter.

```
%%% Tertiary (implementation-level) macro

\def \l@tt@rspace #1\@nd
    {\ifx  \@nd #1\@nd
           \let \n@xt = \relax
     \else
           \p@rtition #1\@nd
           \ifx \h@ad \sp@ce \hsss \h@ad \hsss
           \else
                \h@ad
                \ifx \t@il \@nd \else \hsss \fi
           \fi
           \@x \def \@x \n@xt \@x
                {\@x \l@tt@rspace \t@il \@nd}%
     \fi
     \n@xt
    }
```

The operation of token-list partitioning is conceptually simple: one passes the token list as parameter text to a macro defined to take two parameters, the first simple and the second delimited; the first element of the list will be split off as parameter-1,

and the remaining material passed as parameter-2. Unfortunately this naïve approach fails when the first element is a bare space, as the semantics of TEX prevent such a space from ever being passed as a simple parameter (it could be passed as a delimited parameter, but as one does not know what token follows the space, defining an appropriate delimiter structure would be tricky if not impossible). The technique used here relies upon the adjunct macro `\m@kespacexplicit`, which replaces a leading bare space by a space protected by braces; such spaces may legitimately be passed as simple parameters. Once that operation has been completed, the re-constructed token list is passed to `\p@rtiti@n`, which actually performs the partitioning as above; again `\@x` (`\expandafter`) is used to expand `\b@dy` (the re-constructed token list) into its component elements before being passed as parameter text.

```
%%% Adjunct macros -- list partitioning

\def \p@rtition #1\@nd
    {\m@kespacexplicit #1\@nd
     \@x \p@rtiti@n \b@dy \@nd
    }

\def \p@rtiti@n #1#2\@nd
    {\def \h@ad {#1}%
     \def \t@il {#2}%
    }
```

The operation of making a space explicit relies on prescience: the code needs to know what token starts the token list before it knows how to proceed. Prescience in TEX is usually accomplished by `\futurelet`, and this code is no exception: `\futurelet` here causes `\h@ad` to be `\let` equal to the leading token, and control is then ceded to `\m@kesp@cexplicit`.

The latter compares `\h@ad` with `\sp@cetoken` (remember the convolutions we had to go through to get `\sp@cetoken` correctly defined in the first place), and if they match (i.e. if the leading token is a space), then `\b@dy` (the control sequence through which the results of this operation will be returned) is defined to expand to a protected space (a space surrounded by braces), followed by the remainder of the elements; if they do not match (i.e. if the leading token is *not* a space), then `\b@dy` is simply defined to be the original token list, unmodified. If the leading token *was* a space, it must be replaced by a protected space: this is accomplished by `\pr@tectspace`.

The `\pr@tectspace` macro uses a delimited parameter structure, as do most of the other macros in this suite, but the structure used here is unique, in that the initial delimiter is a space. Thus, when a token-list starting with a space is passed as parameter text, that space is regarded as matching the space in the delimiter structure and removed; the expansion of the macro is therefore the tokens remaining once the leading space has been removed, preceded by a protected space `{ }`.

```
%%% Adjunct macros -- <space>... -> {<space>}...

\def \m@kespacexplicit #1\@nd
    {\futurelet \h@ad \m@kesp@cexplicit #1\@nd}

\def \m@kesp@cexplicit #1\@nd
    {\ifx \h@ad \sp@cetoken
          \@x \def \@x \b@dy
                   \@x {\pr@tectspace #1\@nd}%
     \else
          \def \b@dy {#1}%
     \fi
    }%

\@x \def \@x \pr@tectspace \sp@ce #1\@nd {{ }#1}
```

The final step is to re-instate the category code of commercial-at.

```
%%% re-instate category code of commercial-at

\catcode `\@ = \the \@code
```

Thus letter-spacing is accomplished. The author hopes both that the code will be found useful and that the explanation which accompanies it will be found informative and interesting.

◇ Philip Taylor
The Computer Centre, RHBNC,
University of London, U.K.
<P.Taylor@Vax.Rhbnc.Ac.Uk>