effort studying the results so far and rearrange by hand any items not yet in proper order. Sooner or later, automated activity must end, and some other kind of thought is indicated.

If you prefer the references listed in chronological order, rather than alphabetical order, you might use macro names like `\BJIIbrillhart`, etc., substituting `A, B, ... I, J` for the digits `0, 1, ... 8, 9` in dates of publication (with more such "digits" at the end to cope with authors having more than one item per year). Then the same sorting process may be used to make the first (rough) sort of `bibliog.uns` as before.

Some authors, especially historians, favor endnotes that are much more extensive than mere bibliographical citations; for endnotes of this kind, some of which may consist of several paragraphs (and may contain cross references to one another), the scheme described above is quite inappropriate. Such endnotes are typographically equivalent to solutions for exercises. How to handle solutions for exercises and discursive endnotes are topics for a later tutorial in this series.

In the next episode, code will be described to produce cross references and marginal notes. In particular, we shall give another version of `biblio.set` containing provision for displaying the marginal notes shown in FIGURE 1.

Note. A disk (5.25 DSDD) containing source text for FIGURE 1 and the code files used to produce FIGURE 1 is available for MS-DOS users who are members of the TEX Users Group. In addition, code is included for trapping typographical errors in the bibliographic citations as well as identifying (for the case of alphabetical order) bibliographic items not actually cited. The disk also contains source text, including the code samples displayed, for draft versions of other tutorials in the pipeline for this series. Send $6 (which includes a royalty for the TEX Users Group) to the address below. Outside North America, add $2 for air postage.

It is a pleasure to acknowledge the generous help and encouragement of Barbara Beeton and Ron Whitney, without which these ideas would not have been developed.

◇ Lincoln K. Durst
46 Walnut Road
Barrington, RI 02806

Macros for Indexing and Table-of-Contents Preparation

David Salomon

Introduction

Two macros are presented and described in detail. The first is very useful for the preparation of an index; the second prepares a table of contents (toc). It should be noted that LATEX has macros for similar purposes. Ours, however, are different. Our index macro can produce both silent and non-silent index items, whereas LATEX's only generates silent ones. Our toc macros can easily be modified by the user to specify any format for the table of contents.

Another important aspect of the macros is that they are described in detail, thereby illustrating the concept of a multi-pass TEX job and the use of several advanced TEX features, such as active characters, file input/output, `\edef`, `\futurelet`, and `\expandafter`.

The Index Macro

A good index is important when writing a textbook. So much so that Knuth, on several occasions (see reference 1 pp. 423–425, and reference 6), said that he does not believe in completely automating the preparation of an index, and he always puts the final touches on his indexes by hand. As a result, "*his books tend to be delayed, but the indexes tend to be pretty good.*"

Index preparation by computer is not a simple problem. References 1–2 discuss certain features that a good index should have, and how to incorporate them in TEX. The macro described here is relatively simple (even though some readers may not think so) and implements only one advanced index feature namely, *silent* index entries. However, as an example of a TEX macro it is very interesting because it illustrates the use of the features mentioned above.

The macro accepts an index item and writes it on a file, for the future generation of an index. Its main feature is the use of *optional parameters*. The macro accepts either one, two, or three parameters, of which only one is mandatory. The main parameter should be delimited, as usual, by braces, and the optional ones, by square brackets '`[`' '`]`'. The macro writes all its parameters on the index file, as one string. However, only one parameter, the main one, is typeset. The optional parameters are treated as silent index items, items that should appear in the index but not in the text itself. A good example is a sentence such as:

```
The late Dr.~Mad used to say: ``Computers
  are good, only people are bad.''
```

When writing such a sentence, the author might want to generate the three index items:

```
Mad Nick, 1923--1987
Quotations---Computers
Mad---quotations
```

This is why a good index macro should support silent parameters. We selected the question mark '?' as the name of our macro (see below), so the sentence above should be typed:

```
The late Dr.~?{Mad}[Nick, 1923--1987] used
to say?[Mad---Quotations]{}:
``?[Quotations---]{Computers} are good,
only people are bad.''
```

If the optional parameters are used, one of them should precede, and the other one follow, the main parameter. All the parameters are written on the index file — with spaces separating — as one string, followed by the page number. It should be noted that many LaTeX macros support optional parameters.

Examples of the use of the index macro are:

```
?{...}
?[...]{...}
?{...}[...]
?[...]{...}[...]
?[...]{}
?{}[...]
```

Note that the main parameter, in braces, should always be present; even if it is empty, as in the last two examples. This happens when the entire index item should be silent, as in `?[Mad---Quotations]{}` above.

The main problems in writing this macro are:

1. The macro name should be as short as possible — since it is going to be used a lot — and we have selected the question mark '?' as the name of the index macro. Short macro names consist of a backslash followed by one character. However, it is possible to declare a character as a macro name — by declaring it an *active character* — and then the '\' is not necessary. The character '?' is thus declared active by ``\catcode`?=\active``. Since we still want to be able to typeset a question mark, we define a control sequence `\?` as the ASCII code of '?' by ``\chardef\?=`\?``. There is, of course, nothing special about the question mark. Any other character can be used as the name of the index macro. Reference 1 (p. 423) uses the circumflex '^', but, since the circumflex is also used in math mode for a superscript, care should be taken not to mix the two uses.

2. The macro should be able to take 1, 2, or 3 parameters. This is achieved by writing several macros that examine the next character in the text and, if it is a '[', treat it as the start of another parameter, collect the rest of that parameter, and save it (it is saved in a macro called `\save`). The saved text is later written on the index file, together with the rest of the string. This part of the macro uses the `\futurelet` control sequence, and is described below.

3. Several index items may be declared on a single page, and their optional parameters should all be saved, as described above. Since our macro always saves text in the same place (in macro `\save`), we should write the saved text onto the index file *immediately*. This is usually accomplished by `\immediate\write`, which writes the saved string on the index file immediately, so the next string can be saved in the same place.

In our case, however, the actual writing on the index file must be deferred, since we want to include the page number with each index item, and this number is only known in the output routine. Our macro should, therefore, use `\write` instead of `\immediate\write`. The problem is that, by the time we get to the output routine, several strings, from several index items, may have to be saved. We should, therefore, make sure that macro `\save` is emptied, and its contents written somewhere, before we use it again, for the next item.

This problem is solved by using a combination of `\write` and `\expandafter`. The `\expandafter` makes sure that the saved text is expanded into the `\write` immediately. The `\write` itself, however, is executed later, in the output routine.

Listing of the macro

Here is a complete listing of the macro:

```
\newwrite\inx
\immediate\openout\inx=\jobname.idx
\def\wrx{\write\inx}
\def\space{ }
\chardef\?=`\? % Define \? as a cs whose
%           value is the ASCII code of ?.
\catcode`?=\active % Now change the
%           cat. code of `?' to 13.
\def?{\futurelet\new\macA} % This is the
%           new definition of `?'.
\def\macA{\ifx\new[\let\next=\caseA
  \else\let\next=\caseB \fi\next}
\def\caseA[#1]#2{#2\def\save{#1 #2}
```

```
14.   \futurelet\new\macB}
15. \def\caseB#1{#1\def\save{#1}
16.   \futurelet\new\macB}
17. \def\macB{\ifx\new[\let\next=\caseC
18.   \else\let\next=\caseAB\fi\next}
19. \def\caseC[#1]{%
20.   \wrx\expandafter{\save \space#1---page
21.   \folio;}}
22. \def\caseAB{%
23.   \wrx\expandafter{\save---page \folio;}}
```

Lines 1–2 are concerned with the declaration, naming, and opening of the output file. The file gets the internal name `\inx` and the external name `\jobname.idx` (`\jobname` is a control sequence whose value is the name of the source file being processed by TeX). Lines 5–8 define the control sequence `\?` and declare '`?`' as an active character. Lines 9–18 are concerned with the optional parameters (problem 2 above), and lines 3–4, 19–23 deal with problem 3 above.

Line 9 is the definition of macro '`?`'. It uses the `\futurelet` control sequence, which will be briefly explained. When TeX sees the following:

`\futurelet` ⟨*token1*⟩⟨*token2*⟩⟨*token3*⟩

it:

1. Lets ⟨*token1*⟩ be = the (future) ⟨*token3*⟩.
2. Proceeds as usual, scanning and executing ⟨*token2*⟩, ⟨*token3*⟩, ...

Reference 4 is a tutorial on `\futurelet`, describing an example similar to ours.

In the case of macro '`?`', on line 9, ⟨*token1*⟩ is `\new` (the name of a control sequence, initially undefined), ⟨*token2*⟩ is the macro `\macA`, and ⟨*token3*⟩ is the first token following `?` in the input stream. Note that, on line 9, when `?` is defined, ⟨*token3*⟩ is unknown. It only becomes known after `?` is expanded. Since ⟨*token3*⟩ is the first token following `?`, it is usually a '`{`' (which starts the main parameter). However, if the first parameter is optional, ⟨*token3*⟩ is a '`[`'. As a result, macro `\new` is `\let` to either a '`{`' or a '`[`', and macro `\macA` is expanded.

`\macA`, in turn, examines `\new` and, based on its value, expands either macro `\caseA` or macro `\caseB`. Macro `\caseA` thus handles the case where the first parameter starts with a '`[`' (is silent). The macro therefore assumes that the parameter list has the form `[...]{...}` followed, perhaps, by a third, optional, parameter. The macro typesets the second (non-silent) parameter and saves both parameters. It then expands macro `\macB` to find out if the next token is a '`[`' (i.e., if there is a third parameter). Macro `\macB` compares the next token to a '`[`' and, if they are equal, expands `\caseC`; otherwise, `\caseAB` is expanded. Macro `\caseC` thus handles the case where the macro has all three parameters. Using `\expandafter`, it immediately expands the saved material (the first two parameters), adds the third parameter and the page number, and executes a deferred `\write`. Macro `\caseAB` is similar, only it does not add any third parameter.

Macro `\caseB` is expanded if the first parameter does not start with a '`[`'. In such a case, the macro may have either one parameter or two (with the second one silent). `\caseB` therefore saves the first parameter and expands `\macB` to determine if there is a second one.

The construct

`\wrx\expandafter{\save---page \folio;}`

on lines 20, 23, is a simple application of `\expandafter` to reverse the order of expansion of the two tokens '`{`' and `\save`. When TeX reads the `\write\inx` command, it expects to find a '`{`' followed by the material to be written on the file. TeX is ready to save that material—as a whatsit—on the main vertical list, for a delayed write (see ref. 1, p. 227). However, in our case, TeX finds the `\expandafter` instead of a '`{`'. This causes TeX to look at token `\save` first (to expand it), and then to place token '`{`' in front of it. The result is to force TeX to expand `\save` immediately (to empty it), even though the `\write` is done later. Macro `\save` is now ready for the next index item.

Complicated! But this is the nature of advanced stuff.

The same method is used in this article later, to solve the same problem when preparing a table of contents.

Tests

To test the index macro, we adopt an idea of Winograd and Paxton (ref. 2), and define a macro, `\setpage`, to simulate a page break, so a small amount of text can be spread over several pages.

`\def\setpage#1 {\par\vfill\eject\pageno=#1}`

The text in Figure 1a was used to test the macro. This text generated the index file shown in Figure 1b.

Table-of-Contents Generation

A table of contents should contain an item for each chapter, section, subsection, etc. To generate these items automatically, we need to define macros that will be expanded at the beginning of each chapter, section, etc. Such a macro has two functions—it

```
\setpage 3
Because ?{token matching}[in associative
memory] is so appropriate for an
?{associative memory}[---use for token
matching], we conclude
\setpage 4
Another common mistake is to include
several ?[token---]{tag fields in} a single
token. This habit, seemingly harmless,
can cause conflicts in ?{token
matching}[---conflicts].
\setpage 7
Fig. 12--27b shows how this approach is
implemented. Each iteration increments
the ?[incrementing the token]{label}[in a
data flow computer], thereby generating
?{unique tokens}[---generation of].
\setpage 12
We conclude that ?[token]{matching} must
involve the ?[token matching
by]{iteration number}[and destination],
and a general way of achieving this is
to add a new field, a ?{label}[in data
flow tokens] field, to each token, and to
match tokens by destination, handedness,
{\it and} label.
\setpage 13
A token $A_i$, arriving at the ?{matching
store}[---token arrival at] will match
itself to a $B$ token having the same
label. Such a token is $B_i$ (a $B$ token
that was generated by the same iteration
as $A_i$).
?[matching tokens---various methods]{}

There is another approach to the same
problem. The matching store can be
redesigned such that matching is done
?[token matching]{in order of
seniority.}[A different approach] A token
$A_i$ will always match itself to the
oldest token which has the same
?{destination}[and handedness, used in token
matching] and opposite handedness.
\setpage 16
In our case this would mean that $A_i$
would match itself to $B_i$. In other cases,
however, matching by seniority does not work.

Problem: What could be a good ?{data
structure}[for matching by seniority]
for such a matching store\?
```

Fig. 1a

should typeset the name and number of the section, taking care of vertical spacing and pagination—and it should make sure that the proper toc information goes on an auxiliary file, named `\jobname.toc`, for a later preparation of the table of contents. The macros defined here are called `\chapter`, `\section`, `\ssection`. Other macros, such as for a sub-sub-section, can be defined along the same lines.

The `.toc` file should be `\input` in subsequent passes and used to prepare the final table of contents. Our approach to this problem is to write only the basic toc information on the file, then to create the table of contents by the single command `\input\jobname.toc`. This way, the formatting of the table of contents is separated from the problem of preparing the `.toc` file. To change the format of the table of contents, one only needs to change the definitions of the formatting macros, without changing the format of the `.toc` file.

An important feature of this example is that it requires a number of passes over the source file. Traditionally, the pages of a table of contents are numbered separately, using roman numerals. If, however, we want the toc pages to be numbered as part of the document (i.e., to have page numbers $1, 2, \ldots$), we have to perform two passes. In the first pass the `.toc` file is generated and, at the start of the second pass, it is `\input`, and the toc is typeset on the first pages of the document.

A little thinking shows that even two-passes are usually not enough, and a third pass may be necessary. When the toc is typeset, in the second pass, it occupies the first couple of pages, causing the start of the text to be moved from page 1 to an unknown page. The toc just typeset thus has the wrong page numbers. The second pass, however, also creates an auxiliary `.toc` file, this time with the right page numbers. Finally, the third pass typesets a toc with the right page numbers.

Exercise 1: Is a fourth pass ever necessary?

A general problem in any multi-pass job is, how does TEX know which pass it is doing? A simple, elegant, solution is to have all the passes identical. This way there is no need to tell TEX which pass it is currently working on. Our macros therefore start by trying (on lines 1–2 below) to input a `.toc` file. If such a file exists, it is used, on line 6, to typeset the table of contents, otherwise, a message is sent to the log file, to alert the user. On line 8, the file is closed. The last step is to open a new `.toc` file, on lines 10–11.

The steps described so far are accomplished by:

```
token matching in associative memory---page 3;
associative memory ---use for token matching---page 3;
token--- tag fields in---page 4;
token matching ---conflicts---page 4;
incrementing the token label in a data flow computer---page 7;
unique tokens ---generation of---page 7;
token matching---page 12;
token matching by iteration number and destination---page 12;
label in data flow tokens---page 12;
matching store ---token arrival at---page 13;
matching tokens---various methods---page 13;
token matching in order of seniority. A different approach---page 13;
destination and handedness, used in token matching---page 13;
data structure for matching by seniority---page 16;
```

Fig. 1b

```
\newread\toc
\immediate\openin\toc=\jobname.toc
\ifeof\toc
 \message{! No file \jobname.toc;}
\else
 \tochead \input\jobname.toc \vfill\eject
\fi
\immediate\closein\toc
%
\newwrite\toc
\immediate\openout\toc=\jobname.toc
```

The .toc file format

The .toc file described here contains, for each toc item, a simple record with the following fields:

- One of the codes \ch, \se, \sbs, for a chapter, section, and subsection, respectively.
- The chapter (or section) number, followed by a colon ':'.
- The chapter (or section) name, followed by the word '\page'.
- The page number, followed by a '\\'.

A typical, simple .toc file may look like the example below:

```
\ch1:Introduction\page3\\
\se1.1:The Use of Tags\page4\\
\sbs1.1.1:Incrementing the Tags\page7\\
\se1.2:Label of Tokens\page12\\
\sbs1.2.1:Seniority Matching\page13\\
\se1.3:Summary\page16\\
```

Such records are easy to write on the file, and they make it possible to typeset the entire table of contents by the single line

```
\tochead \input\jobname.toc \vfill\eject
```

This is achieved by defining macros \ch, \se, \sbs, to typeset lines in the toc. Macro \ch, for example, typesets a chapter line in the toc. It is expanded automatically during the \input, each time a record starting with a \ch is read off the file. Macros \se, \sbs behave similarly. These macros (plus \tochead, which typesets the heading of the table of contents) are the only ones that typeset the toc and, as a result, the only ones that need to be modified when a different toc format is required. The following are guidelines for writing these macros:

\def\tochead{⟨*typeset a heading for the table of contents*⟩}
\def\ch#1:#2\page#3\\{⟨*typeset a line in the toc, with* #1 *as the chapter number,* #2 *as the chapter name, and* #3 *as the page number*⟩}
\def\se#1:#2\page#3\\{⟨*similarly for a section*⟩}
\def\sbs#1:#2\page#3\\{⟨*similarly for a subsection*⟩}

Writing the .toc file

At the start of each chapter, the user expands macro \chapter with one argument, the chapter name.

```
\newcount\chnum \chnum=0
\newcount\snum \newcount\sbsnum
\def\chapter#1{\global\advance\chnum 1
 \global\snum=0 \sbsnum=0
 ⟨select a font and typeset \the\chnum and #1⟩
 \edef\save{\string\ch\the\chnum:#1%
  \string\page\noexpand\folio\string\\}%
 \write\toc\expandafter{\save}}
```

The macro should typeset the chapter name and take care of vertical spacing and page breaks. Its last step is to store the information necessary for the toc in a macro called `\save`, and to write `\save` on the `.toc` file.

At the start of each section or subsection, the user similarly expands `\section` or `\ssection`, which behave similarly to `\chapter`.

```
9.  \def\section#1{%
10.   \global\advance\snum 1 \sbsnum=0
11.   ⟨typeset \the\chnum.\the\snum and #1⟩
12.   \edef\save{%
13.     \string\se\the\chnum.\the\snum:%
14.     #1\string\page\noexpand\folio
15.     \string\\}%
16.   \write\toc\expandafter{\save}}
17.
18. \def\ssection#1{%
19.   \global\advance\sbsnum by 1
20.   ⟨typeset \the\chnum.\the\snum.\the\sbsnum
        and #1⟩
21.   \edef\save{%
22.     \string\sbs
23.       \the\chnum.\the\snum.\the\sbsnum:%
24.     #1\string\page\noexpand\folio
25.     \string\\}%
26.   \write\toc\expandafter{\save}}
```

The macros have to deal with two related problems, namely the chapter number and the page number on the toc file.

The page number, `\folio`, is not known when the toc record is created. It only becomes known when the output routine is invoked. The `\write` should therefore be delayed. This is a common problem and is solved simply by saying `\write` instead of `\immediate\write`.

The chapter number, `\the\chnum`, on the other hand, is known and should be expanded immediately. If its expansion is delayed to the output routine, the number expanded will be the chapter number in effect during the output routine. The same applies to the section and subsection numbers.

These problems are solved, in macro `\chapter`, on lines 6–8.

Lines 6–7 define macro `\save` with the necessary information for a single toc record. The `\edef` control sequence is used, instead of `\def`, to guarantee that the chapter number, `\the\chnum`, that is expanded inside `\save` will be the one in effect when `\save` is *defined*, not the one when `\save` is expanded.

The `\noexpand\folio`, on the other hand, guarantees that `\folio` will not be expanded when `\save` is defined; instead, it will be expanded when the `\write` is expanded (in the output routine).

The use of `\expandafter` has been explained earlier, in connection with index preparation.

Exercise 2: Experiment with lines 6–8 above to find out what happens when the `\edef` is changed to `\def`, when the `\noexpand` is omitted, and when the `\expandafter` is dropped.

Limitations

1. The size of a record on a file is limited by the operating system of the computer. Since each line of the table of contents goes on the file as a record, its size is limited and, as a result, we cannot have chapter or section names which are too long. Note that this limitation has nothing to do with TeX.
2. When a large document—such as a book—is typeset, it is common to typeset each chapter individually, creating its own `.toc` file. In such a case it is possible to create the final table of contents by a special TeX job. Each of the individual `.toc` files is input, and the table of contents is numbered separately, using roman numerals. This only requires two passes, and generates a toc similar to the traditional method.

Exercise 3: Sometimes the book designer wants the chapter numbers in roman numerals, how can this be done?

Acknowledgement

The author would like to thank Ron Whitney for his many important comments and suggestions. They have caused a major revision of this work, and have made it much more useful to the readers.

Answers to exercises

1. Yes, if the text is modified in any of the passes. Even more than four passes may be necessary in such a case.

2. Just do the experiments.

3. Use

```
\uppercase\expandafter
  {\romannumeral\the\chnum}
```

instead of `\the\chnum` on line 3 in macro `\chapter`. This tricky construct is demonstrated in exercise 7.9 of ref. 1.

References

1. Knuth D. E., *The TeXbook*, Addison-Wesley, Reading MA: 1987.

2. Winograd, T. & B. Paxton, *An Indexing Facility for TEX*, *TUGboat* **1**(1) (Appendix A).
3. Chen, P. & M. A. Harrison, *Index Preparation and Processing.* Soft. Practice & Exp. **18**(9), 897–915 (Sept. 1988).
4. Bechtolsheim, S., *A Tutorial on* \futurelet, *TUGboat* **9**(3), 276–279, 1988.
5. Bechtolsheim, S., *A Tutorial on* \expandafter, *TUGboat* **9**(1), 57–61, 1988.
6. Knuth D. E., *Typesetting Concrete Mathematics*, *TUGboat* **10**(1), 31–36, 1989.

◇ David Salomon
California State University, Northridge
Computer Science Department
Northridge, CA 91330
bccscdxs@csunb.csun.edu

Query

Editor's note: When answering a query, please send a copy of your answer to the *TUGboat* editor as well as to the author of the query. Answers will be published in the next issue of *TUGboat* following their receipt.

A Scribe-to-TEX Converter

One of the SEMATECH consortium members donated a software product with lots of documentation. Unfortunately it's marked-up using Scribe. SEMATECH has modified the software product to meet our needs, but the prospect of un-SCRIBE-ing and then TEX-ing hundreds of large user and system documentation files with a text editor is not attractive. Please let us know if you are aware of any Scribe-to-TEX translators.

Terry Bush
SEMATECH
Montopolis Research Center
University of Texas
2706 Montopolis Drive
Austin, Texas 78741
(512) 356-3443
terry_bush@sematech.mrc.utexas.edu

LATEX

Towards LATEX 2.10

Frank Mittelbach and Rainer Schöpf

After the TUG meeting at Stanford, Leslie Lamport expressed interest in future developments of LATEX. He and one of the authors (FMi) agreed on a two-stage procedure for this [1, 2]. The first step will be a new style file interface. Therefore we are interested in any style file which implements features that are not provided in the current document styles.

Independently of these efforts we are planning to publish the implementation of a number of enhancements to the current LATEX version:

- A new `verbatim` environment.
  This includes a `\verbatimfile` command to read in a file of verbatim text, and a `comment` environment that discards all text in its body. Other features are: no limitation on the size of the verbatim text, and the possibility of using `verbatim` inside other environments.
- A new version of the doc–option.
  One of the most important improvements over the version published in the previous issue of *TUGboat* is the introduction of a check to detect truncations during transmission. We are very interested in hearing about experiences other people have had with this style option. Suggestions for improvements are welcome.
- Enhancements to the new `array` and `tabular` environments published in *TUGboat* 9#3.
  Again, suggestions are welcome.
- The interface between LATEX and the new font selection scheme.
  This interface consists of two parts: one emulates the font selection mechanism of standard LATEX and is ready to use. The second part is made to give full control over the new scheme. However, field tests have shown that the commands we provided for this are not user friendly enough to be released yet.

We are sorry that we have to report a small but very important typo in the article on the new font selection scheme (*TUGboat* 10#2, pp. 222–238).[1] It is very important because it is in the code, namely in the macro `\mathversion` (p. 230): in the first line of the macro definition the primitive `\endcsname` is erroneously spelled "`\endscname`".

[1] Thanks to Sebastian Rahtz for finding this one.