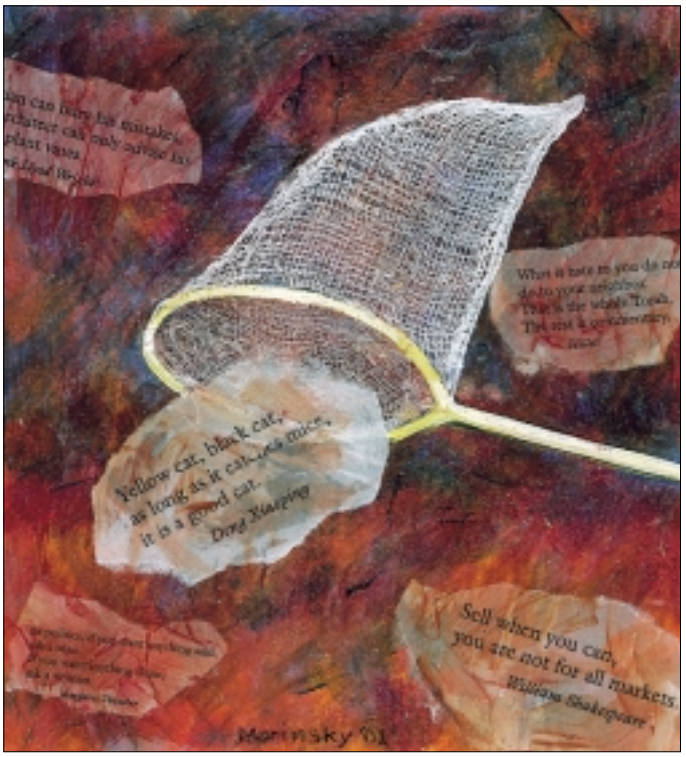
by Jeffreys Copeland and Haemer



JANE MARINSKY

It is necessary that practice accompany knowledge.
– Gottfried Wilhelm Leibnitz

Ah, it's a lovely thing, to know a thing or two.
– Moliére

Commonplace Book

Back in our grandfathers' day, a well-read gentleman always kept a "commonplace book" in which he copied out interesting passages he found in the various things he'd read. He could find them again later and share them with his children and grandchildren.

In our modern age, this is a habit that's nearly fallen by the wayside. A pale reflection survives in the habit of some folks who post lists of interesting quotations on their Web sites. We keep an online file of quotations, too, some of which you have seen as epigrams at the top of this column. However, we're sometimes dead tree guys. While we appreciate the electronic forms of things, as you know from reading last month's column on preparing text for an electronic reader, occasionally we want to have a nice printed thing on paper to curl up with in our wing-back chairs and hold in our hands. So this month, we're going to talk about turning your online collection of quotations into paper, complete with nice formatting and index. We'll get to explore some subtleties of formatting with TeX, and some tricks for making an index. We've got three distinct sub-problems to solve: formatting the quotations, preparing the index as we go, and printing the index from the "notes" we've taken. If you're not familiar with TeX, we point you at the original reference by Donald Knuth, *The TeXbook*, (Addison-Wesley, 1984, ISBN 0-201-13448-9). Let's dig in.

A Basic Quotation

Let's begin with the basic form of a quotation in our file.

```
\quote{I wondered what a savoury
scandal would be: a scandal
fried on toast, perhaps, with an
anchovy and a dash of Worcester
Sauce?}{\q{Rumpole and the Case
of Identity} by John Mortimer}
in \em{The Trials of Rumpole},
1979.
```

We've postulated a two argument TeX macro `\quote` which takes the quotation and the attribution. We've even added a little notation after the quotation–sometimes it will be a witty aside. We choose to do this in TeX, rather than `troff`, because some of the indexing turns out to be easier. Those of you fluent in both text formatting languages can follow along and translate as you go. For example, if we were going to do this in `troff`, we'd have a macro to start the quotation, one to signal the attribution start, and one to complete it. We'd say something like:

```
.Quote
I wondered ...
.Quote-attr
''Rumpole and ...
.Quote-end
```

What's the definition of our `\quote` macro?

```
\long\def\quote#1#2{
  \filbreak\bigskip
  \noindent
  \hrule height 1pt\smallskip
```

```
  #1\par
  \def\z{#2}
  \ifx\z\empty \else
  {\leftskip .3\hsize\relax
  \item{—-}#2\par}\fi
  \smallskip\noindent\hrule
  \smallskip\noindent
 }
\def\empty{}
```

First, notice that it is a `long` definition, so that our quotation can include a paragraph break. We begin with a `filbreak`. If you're not familiar with it–and it's one of our favorites–this macro says "here's a good place to break the page, but do not bother if you can get up to the next `filbreak` on this page, too." This means that quotations, unless they're more than a page long, will be on a single page. We offset the quote with a one point rule–slightly thicker than the default–then set the first argument (the quotation itself). If the second argument is not empty, we set it with a large indent preceeded by an em-dash. We finish up with a less-heavy rule at the default thickness of 0.4 points. Our extra notations appear after the closing rule, in the space before the next quotation.

Why the hand-wave of assigning the second argument to `\z`? Because we want to test the expanded values of the second argument and `\empty` with `\ifx`, and to do that they have to be at the same level of reference.

Notice that rather than a double quotation mark around the title of the Rumpole story in our example, we've used a `\q{...}` macro. This allows us to nest quotes easily. Similarly, we'll be using a `\em{...}` macro (the idea for which we stole from LaTeX) to set emphasized text, such as titles, usually in italics. By making features of the structural markup–quote and attribution, quotation marks, titles–into macros, we can change the rendition of our printed version much more easily. As a quick example, if we wanted to produce a British version, in which the outer quotation marks were single and the inner quotation marks were double–'She asked "Why did you only shoot him seven times?" with a tear in her eye.'–this would involve a quick rework of the macros, not a complete edit. The quotation mark macros have some interesting features:

```
%% quotation mark macro
\newcount\qm \qm=0
\def\q#1{\advance\qm by 1
 \ifodd\qm "#1"\else
  '#1'\thinspace\fi
 \advance\qm by -1\relax}
```

We increase the counter of active quotation marks every time we see the macro. If we have an odd number, we use double quotation marks; an even number produces singles. This allows nested quotation marks through a nested use of the macro to work as well. We have to be careful to not let any white space drift in after the quotation marks, so that

```
\q{foo}—-he exclaimed.
```

will not have spurious interword spacing. However, we need the `\thinspace` in the single-quote case to properly space when we want a single quote followed by a double. The inverse case–double followed by single–is handled by TeX's normal kerning rules.

After that, the emphasis macro is downright simple:

```
\def\em#1{{\it #1}}
```

Embedding Index Entries

It makes this collection of quotations much more useful if we can provide an index of the sources. It would be useful to be able to find all the references to barrister Horace Rumpole, all the passages from *Molly Ivins Can't Say That, Can She?*, or all the Donald Knuth quotations we have.

It will serve us well to think for a few moments about the design before we delve in. We will want to index names and titles in different fonts. Even among the names, it would be nice to index character and author names, and perhaps even actor names if we've been collecting movie quotes, differently. So we'll want not only a name, but a tag, so that we can distinguish the fictional character Rumpole, Horace from the writer Mortimer, John.

Let's try this:

```
\idxi{The Trials of Rumpole}
\idx{Mortimer, John}
\idxc{Rumpole, Horace}
```

What happens to these additions under the covers? We add a one-character tag, as we discussed, to each one and pass it on to another macro:

```
\def\idx#1{\IDX{#1}{n}}
\def\idxc#1{\IDX{#1}{c}}
\def\idxi#1{\IDX{#1}{i}}
```

You can envision generalizing this scheme, we're sure, not only for actors' names as we mentioned above, but also to provide entries in a typewriter-like font for entries such as `rec.humor.funny` or `jsh@usenix.org`.

The underlying (and more general) `IDX` macro presumes a file into which we write the index entries, so we name and open it first.

```
\newwrite\idxfile
\openout\idxfile=quotes.idx
\def\IDX#1#2{\write\idxfile{#1;#2;\the\pageno}
}
```

We don't have to worry about the page number on which the index entry appears because the `filbreak` in the `quote` macro ensures that the end of the quotation is almost always on the same page as the beginning.

So far, this has been simple, but wouldn't it be much easier if we could get most index entries directly out of the text? Yes, we'll still be able to use the `{co \idx}` macros but what if we

could make a macro that both typeset the argument and put it in the index? Something like:

```
\quote{ ...  It was, carried to the extreme,
as though someone had put a Brooks Brothers
suit on a gorilla.}{\!Naked Came the
Stranger!, by Penelope Ashe}
```

That turns out to be pretty simple. All we have to do is add another wrapper macro like

```
\def\!#1!{\textset\em{#1}\endgroup\idxi{#1}}
```

In this, the `textset...endgroup` surrounds our index entry when we set it as part of the body text. To see why, we have to consider how we'd index "Penelope Ashe" out of the running text. Clearly, we want to make an index entry for "Ashe, Penelope" from this. If we used `\( and)` as delimiters and encoded this as

```
\(Penelope\\Ashe)
```

we'd be halfway there. We could use the double backslash and closing parenthesis as delimeters to the macro. In running text, the double backslash would turn into a space; for

Wouldn't it be much easier if we could get most index entries directly out of the text?

the index entry, it would be a marker to invert the names at this point. That scheme falls apart if we want a name index entry that we want in multiple places, such as

```
\(HRH\\Charles Philip Arthur
George, the\\Prince of Wales)
```

which we want to index under both "Charles ..." and "Prince of Wales." Remember that we're going to have to alphabetize the index anyway, so if we just copy the double backslash to the index file, we can deal with inverting the names later. We insert the double backslash in the index with the following macro snippet:

```
\chardef\SP='\\   \let\\=\SP
\def\textset{\begingroup\def\\{ }}
```

We begin by defining the character `SP` to be double backslash, and then letting the macro `\\` stand in for that definition. The net effect is when this definition is active, the argument-less double backslash macro is rendered as itself. In the `textset` macro, on the other hand, the `begingroup` pushes the context, so that when we reach an `endgroup`, the original context–with the original definition–returns. The one odd note is that the "normal" definition of double backslash is active when the index entry is being written into its file; the definition in the alternate context inside the `begingroup` is used for the running text. Thus, we can add a definition

```
\def\(#1){\textset{#1}\endgroup\idx{#1}}
```

which will turn our Ashe entry above into "Penelope Ashe" in the running text, and "Penelope\\Ashe" in the index file.

However, there is another wrinkle: In the text, we will often want to credit "P. J. O'Rourke" and then in the index refer to "Peter John O'Rourke." We do this with an extension of the trick we just used, and invent the `\[...]` macro.

```
\def\[#1]{#1}
```

We need to change our definition of textset to

```
\def\textset{\begingroup
    \def\[##1]{\relax}
    \def\\{ }}
```

(We'll pause for a moment to air some dirty laundry: The Jeffreys have an ongoing battle about the use of periods in initials and abbreviations. Haemer prefers the strictly American approach of always following initials and abbreviations with a period. He's willing, however, to tolerate the British preference for periods after initials but eliding them after abbreviations that end with the final letter of the word being abbreviated. Thus, "Mr." in American, but "Mr" in British. Copeland prefers the more idiosyncratic usage of eliding the period in all cases, as in "Mr P J O'Rourke." The macro definition above is arranged for Copeland's preference. We leave it as an exercise to the reader to rearrange the macros to substitute a period as appropriate. Remember that sometimes we'll want to have an entry like

```
\(Groucho\[(Julius)]\\Marx)
```

where the elided text won't be the back half of an abbreviation.)

Given these steps, you can generalize the indexing process and provide index entries for typewriter-like fonts (as we discussed before), variations for characters and actors, and so on. We've provided our versions of those extras in the software bundle at the usual Web sites, listed at the end of the column.

When we run the file we've prepared this way through TeX, we'll be left with an auxillary file `quotes.idx` containing lines such as

```
The Soul of a New Machine;i;1
Alistair\\MacLean;n;3
Sharyn\\McCrumb;n;23
The Soul of a New Machine;i;29
John\\McMullen;n;73
The New Yorker;i;80
Ulrika\\Anderson\\O'Brien;n;106
```

```
James Abbott McNeill\\Whistler;n;113
Groucho [Julius]\\Marx;n;117
Ulrika\\Anderson\\O'Brien;n;120
The (London) Times;i;129
The (London) Times;i;140
Ulrika\\Anderson\\O'Brien;n;143
2001: A Space Odyssey;i;156
```

We'll want to turn this into entries for the index pages such as

```
\idxp {{\it 2001: A Space
 Odyssey}}{}{156}.
\idxp {Anderson O'Brien, Ulrika}{n}{106,
 120, 143}.
\idxp {Charles Philip Arthur George, the
 Prince of Wales, HRH}{n}{12}.
\idxp {{\it The (London) Times}}{}{129,
 140}.
\idxp {McCrumb, Sharyn}{n}{23}.
\idxp {MacLean, Alistair}{n}{3}.
\idxp {McMullen, John}{n}{73}.
\idxp {Marx, Groucho [Julius]}{n}{117}.
\idxp {{\it The New Yorker}}{}{80}.
\idxp {O'Brien, Ulrika Anderson}{n}{106,
 120, 143}.
\idxp {Prince of Wales, HRH Charles Philip
 Arthur George, the}{n}{12}.
\idxp {{\it The Soul of a New
 Machine}}{}{1, 29}.
\idxp {Whistler, James Abbott
 McNeill}{n}{113}.
```

Notice that we want to alphabetize the entries. We'll even use a librarians' sort, ignoring articles such as "the" and folding "Mc" into "Mac." We also need to collect all the disparate pages on which a name appears into a single entry. Other than noting that we need to actually run TeX twice–once to generate the index data, and then once with that data sorted and marked up, how do we do it?

Preparing the Index

Fred Brooks says "Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious." Or as Elizabeth Schwarzin used to put it, "The secret of life is data structures." Thus, two words should tell you everything that is about to come: "Perl hash."

If we grab each line in turn and make a hash entry for all the variations of names on the line, we'll end up with the list we want. Sort of.

```
while( <> ) {
 chomp;
 ($name,$type,$page) = split /;/;
```

(Because of space considerations, we're not going to show you the usual set up and declarations. Take these as read, and pick up the full script from one of the Web sites.)

Once we've got the name, tag and page number, it's simple to store them away. We can even make a subroutine to do it for us:

```
sub makeentry() {
  my ($name, $type, $page) = @_;
  $name .= $type;

 if( exists $entries{$name} ) {
  $entries{$name} .= " , " . $page;
 } else {
  $entries{$name} = $page;
 }
}
```

Notice that we're just tacking the type onto the end of the name. This means that we can distinguish between an entry where a writer and a character in a book have the same name. Also notice that if the hash already exists, we're appending the page number to it. Since the raw index input is presented in the same order as the quotations themselves–that is, in page order–we end up with the page list already sorted.

Except, it's not that simple. If we have a raw index entry like:

```
Ulrika\\Anderson\\O'Brien;n;106
```

we have to invert the names. Twice. So we need a little wrapper code to call the `makeentry` subroutine.

```
  @names = split /\\\\/, $name;

 if( @names > 1 ) {
  # we need to reverse the subfields
  #   of the index entry
  $opt_tail =
   ($names[-1] =~ s/(.*)(, \w+)/$1/) ?
       $2 : "";
  $names[-1] =~ s/$/,/;
  foreach (2..@names) {
   push(@names, shift(@names));
   $name = join(" ", @names) . $opt_tail;
   makeentry($name,$type,$page);
  }
 } else {
  # basic, simple output
  makeentry($name,$type,$page);
 }
}
```

The simple case–we have no double backslash–is handled in the `else` clause. The more complicated case is handled in the `then`: We take the names, split at the double backslash, and shift through them, making a hash entry for each rotation, that is, for both "Anderson O'Brien, Ulrika" and "O'Brien, Ulrika Anderson." In the simple version, that inner `for` loop will only execute once and we'll end up with `John\\McMullen` becoming the single hash key "McMullen, John."

What's the `opt_tail` all about? We'll occasionally have a name with an ending that we don't want to invert, such as, `\(Jeffrey S\[herman]\\Haemer, PhD)`. Any appendage delimited by a comma should remain at the end: "Haemer, Jeffrey Sherman, PhD."

Sorting the Results

Oddly enough, that was the easier part of the task. We can't just print the hash entries in their natural order, that is, in no order at all. We have to provide some sort of sorting for them. For this, we turn to a trick invented by Randall Schwartz, which has come to be known as the Schwartzian Transformation. (See recipe 4.16 in Tom Christiansen and Nat Torkington's *The Perl Cookbook*, O'Reilly, 1998, ISBN 1-56592-243-3, for an in-depth discussion of this trick.)

In the normal course of events, we can produce a hash in sorted order with a loop of the form:

```
foreach (sort keys %hash) {
 ....
}
```

The normal `{$a cmp $b}` comparison in the sort is implied. This time, though, we have a more complicated sorting we want to accomplish. We want to compare with case folding, eliding articles, and with Scots name conversion (folding together "Mc" and "Mac"). To do this, we can `map` the data beforehand.

A Perl `map` takes an array as input, transforms the data through a little program, and produces another array as output. For example:

```
my @keys = map {
 s/\b(the|a|an)\s//gi;
} keys %entries;
```

will give us an array `keys` composed of the article-stripped keys of my hash. I can then sort this array, and then ... Oops! I've lost information by stripping the articles, and cannot recover the original keys. The solution is to make an array of anonymous arrays:

```
 my @keys = map {
 (my $x = $_) =~ s/\b(the|a|an)\s//gi;
 [ $x, $_ ]
} keys %entries;
```

which produces an array of arrays containing the sortable key and the original hash key. After the sort, I strip off the sortable key, leaving only the hash key with a new `map` transform,

```
my @keys = map { $_->[1] };
```

We combine this into a single statement for the case-in-point:

```
my @keylist =
 map { $_->[1] }
 sort { $a->[0] cmp $b->[0] }
 map { (my $x = $_) =~ s/\b(the|a|an)\s//gi;
  $x =~ s/\bMc/Mac/gi; # Scots names
  $x =~ s/[^\w\s]//g; # elide non-alphanums
  [ uc($x), $_ ] }
 keys %entries;
```

Once we've got a sorted list of the keys, we can format the hash array indexed by them.

```
foreach (@keylist) {
 ($name,$type) = /(.*)(.)/;
 if( $type eq "i" ) {
  $type = "";  $name = "{\\it $name}";
 }
 print "\\idxp
{$name}{$type}{$entries{$_}}.\n";
}
```

(We're indexing the hash of the index with the index keys. You can see how confusing it can be by using "index" in different senses in the same sentence.) Remember we have combined the tag into the key, so we separate `$name` and `$type` with a multiple assignment. If the tag is relevant, we strip it, and insert a font change into the name string to be printed. As output we produce a line of text with TeX encoding in place.

Once we encapsulate this Perl code in a script called `DO.idx`, we have to figure out how to format the index on the output side.

Formatting the Index

Producing a pretty index is a fairly simple matter. We finish our file of quotes with an invocation of

```
\indexprint
```

If we haven't run `DO.idx` over the raw index data yet, we want the `indexprint` macro to fail gracefully. If we do have sorted index data, we want it to produce a two-column list of the entries of the `idxp` lines in that file.

We can begin by postulating a simple `idxp`,

```
\def\idxp #1#2#3.{\def\z{#2}
 \hangp\rm #1
 \ifx\z\empty\else\thinspace(#2)\fi,
 #3.\par}
```

We're producing the index line as a hanging paragraph, using a definition of

```
\def\hangp{\par\noindent\hang}
```

We're also checking if the type tag is non-empty before we print it–we may have elided the tag in favor of font change already in `DO.idx`. We need one extra definition,

```
\newread\indextest
```

which will allow us to read the first line of the sorted index file.

Given those, the wrapping in the `indexprint` macro is, again, straightforward.

```
\long\def\indexprint{
 \vfill\eject
 \let\bf=\Ibf \let\rm=\Irm \let\it=\Iit
 \baselineskip=\idxbaseline \rm
 \centerline{{\bf Index}}
 \message{[Index]}
 \input double \raggedright
```

We make this a `long` definition because each index entry we are printing begins a new paragraph. We finish the last page of quotations with `\vfill\eject` so we can begin the index on a fresh page. We set up some fonts–we alias earlier font definitions `Ibf`, `Irm`, and `Iit` to be the current bold, roman, and italic fonts. We similarly reset the line-spacing. (Again for space, we've skipped the font definitions in these macros; there are parallel definitions for the fonts in the running text and index which we swap as appropriate. By carefully encapsulating the font definitions, we can change the look of our printed version by swapping just a few lines. It's why when working with `troff`, you should refer to fonts by position, not name.) We print out a header and a console message. We finish the setup by reading in the macros for two-column output, `double.tex`, and setting the "paragraphs" to be set unjustified (`raggedright`).

We want to check if the index file exists so we can act appropriately if it doesn't. Here's where we use the definition of `indextest` from above:

```
\openin\indextest=quotes.srt
 \read\indextest to \hitreturn
 \ifeof\indextest
  \centerline{{\it No index file available.}}
  \message{[No index file]}
```

We open the sorted file and read the first line into `\hitreturn`. If the read fails, `eof` is set for the file, and we print messages on the page and the console. (In principle, opening a non-existent file should set the end-of-file indicator, but we've seen at least one TeX port in which you have to attempt a read before that actually happens. We've gotten in the habit of writing this code defensively as a result.)

The `else` clause, when the file does exist, is also simple, and finishes up the macro.

```
\else
  \closein\indextest
  \begindoublecolumns
  \input quotes.srt
  \enddoublecolumns
 \fi
}
```

We read the sorted index file `quotes.srt` with an `\input` statement, surrounding it with `begindouble columns` and `enddoublecolumns`. (We don't have time or space to explore the double-column macros. We use essentially a modified version of the ones from the TeX `manmac` package, though. Others exist at the Comprehensive TeX Archive Network, `http://ctan.tug.org`.)

There You Have It

All that remains is to wrap the process in a few lines of shell command, such as,

```
$ tex quotes
$ DO.idx quotes.idx >quotes.srt
$ tex quotes
$ dvips quotes
$ lpr quotes.ps
```

We've covered a lot of ground, developing multiple techniques for burying indexing data in a document, showing how to do a complicated transformation (including sorting) of a file in Perl, and learned some new formatting tricks in TeX. Maybe one of these techniques can be transplanted into a problem sitting on your desk right now. Let us know how you use them.

There's still some room for improvement here. We've talked about distinguishing different types of index entry, but it might also be helpful to incorporate mini-indexes: When we're preparing the main index, it might also be helpful to show the current index entries after each quotation, either in a small font in-line, or in the margins. How would you do this?

Maybe one of these techniques can be transplanted into a problem sitting on your desk right now.

An alternate solution to all of this would be to use a very general system, for example BiBTeX. While BiBTeX is optimized for bibliographies, it could also be used for managing our quotation file. We didn't do this for two reasons: not only don't we like LaTeX, but we always want to talk more about techniques rather than tools.

That's all for this month. Until next time, happy trails.

Jeffrey Copeland *(*`copeland@alumni.caltech.edu`*) is currently living in the Pacific Northwest, where he spends his time writing UNIX software in a large development organization and fighting damp rot.*

Jeffrey S. Haemer *(*`jsh@usenix.org`*) works at Minolta-QMS Inc. in Boulder, CO, building laser printer firmware. Before he worked for QMS, he operated his own consulting firm and did a lot of other things, like everyone else in the software industry.*

Note: The software from this and past Work columns is available at `http://alumni.caltech.edu/~copeland/work` *or alternately at* `ftp://ftp.cpg.com/pub/Work`.