XSLT Exercise 6

The input collection on our DHClass-Hub

You will be working with same digitized XML collection of Emily Dickinson's poems that you worked with in the last assignment, the collection we posted on our DHClass-Hub GitHub. If you need a new copy of the files, please refer to our instructions on the previous exercise for accessing them by syncing, cloning, and copying the directory out of your local GitHub directory. You will be building on our XSLT Exercise 5, and you can take your stylesheet from that assignment and modify it for this one.

Overview of the assignment

For your last assignment you used the XSLT `@mode` attribute to create a table of contents for the Dickinson poems in Fascicle 16, using the poem number and first line of each poem as a surrogate for the title (since they don't have real titles). Our output for that previous assignment is at http://newtfire.org/dh/dickinson-5.html.

What's a table of contents good for anyway?

In a digital edition, we can just do a full-text search and scroll in the browser, so we don't really need a table of contents at all. We can search for a poem by number, we can search for the text of the first line, or we can search for a memorable phrase. But suppose we want to produce a paper edition, where the only organized access our users will get is the organization we decide to give them. What would be a useful table of contents or index?

A table of contents in the same order as the full text (numerical order), which is what we produced in the last assignment, duplicates the ordering information in the plain text. How useful is that? If we want to find a poem with a low number, we already know without a table of contents that we should look near the beginning. On the other hand, it's very common in published poetry collections to include an index of first lines, sorted in alphabetical order, so that a user who remembers just the first line of a poem can find it easily. It is a little less common to sort the order of a series of poems by an interesting feature inside them, but since one of our interests in digitizing this collection is to study Dickinson's use of variants, we would also like to sort the poems by a count of the variants in each, so that the poems with the highest number of variants come first in the sort order.

For this assignment we're going to enhance our output from the last assignment in the following ways:

- We're going to create links between the items in the table of contents and the poems, so that you can click on a poem's identifying information and be taken immediately to the corresponding poem.
- We're going to produce a list of the poems organized by the total number of the variants that Dickinson marked in them, from most variants to least variants.

- We’re then going to alphabetize our list of first lines, so that the table of contents will be sorted alphabetically, instead of in numerical order.

Our HTML output for this assignment is at http://newtfire.org/dh/dickinson-6.html.

The tools we need

To create links between the first lines in the table of contents and the poems in the full text section of the page below we’re going to use *attribute value templates* (AVT). We have been working with these in earlier XSLT assignments, but you may want to review Obdurodon’s page on how they are written: http://dh.obdurodon.org/avt.xhtml.

To sort the table of contents we’re going to use `<xsl:sort>`.

When we sort the first lines, they won’t sort correctly for a quirky reason. We’re going to fix that using the XPath `translate()` function, which we discuss below.

How HTML linking works

The `<li>` items in the table of contents should include `<a>` ("anchor") elements, which is how HTML identifies a clickable link. An anchor that is a clickable link has an `@href` attribute, which points to the target to which you want to move when you click on the link. For example, the table of contents might contain the following list item for Poem 6:

```
<li><a href="#p1606"><strong>Poem 6 (J 281: 1861/1935)</strong></a>:
               1: 'Tis so appalling—it exhilarates— <br /> [Variants: 4]</li>
```

HTML `<a>` elements that have `@href` attributes normally appear blue and underlined in the web browser, to advertise that they are links. The *target* of a link can be any element that has an `@id` attribute that identifies it uniquely. (This is why you need to use a hashtag (#) in the `@href` on the Table of Contents that links to an `@id`, because the # indicates you are pointing to the unique identifier that follows.) If you click on this line in the browser, the window will scroll to the element elsewhere in the document that has an `@id` attribute with the value "p1606". In our case, we’ve assigned that `@id` attribute value to the `<h2>` for that poem in the main body:

```
<h2 id="p1606">Poem 6 </h2>
```

Adding links to your output

You should first review Obdurodon’s page on Attribute value templates (AVT), which describes a strategy you can use to create a unique `@id` attribute for each poem. For this task we gave our poems `@id` values that were a concatenation of the letter "p" and the distinct identifying number given in the `idno` element in the TEI header of each poem file: "1606" for Poem 6. We attached those `@id` attributes to the `<h2>` elements that we used as titles for each poem in the body of our page, e.g., `<h2 id="p1606">`. Meanwhile, in the table of contents at the top we created `<a>` elements with `@href` attributes that point to these `@id` values. *The value of the `@href` attribute must begin with a leading "#" character, but that "#" must not be part of the value of the `@id` attribute to which it points*. For example,

```
<li><a href="#p1606"><strong>Poem 6 (J 281: 1861/1935)</strong></a>:
               1: ’Tis so appalling—it exhilarates— <br /> [Variants: 4]</li>
```

means if the user clicks on the linked content in this list item, the browser will scroll to the line that reads `<h2 id="p1606">` in the main body of the page. *Remember: the value of the `@href` attribute begins with "#", but the value of the corresponding `@id` attribute on the `<h2>` element you want to scroll to doesn't.*

Armed with that information, you can take your answer to the main assignment and, using AVTs, modify it to create the `<a>` elements with the `@href` attributes and the `@id` attributes for the targets.

Sorting

An index of first lines in a collection of poems is usually alphabetized, because that's how humans look things up in that kind of list. We want to make an alphabetized list by first line, as well as sorted list by count of the variant phrases we have marked in these poems, so we wish to do two kinds of sorting in this assignment: one that is alphabetical and the other based on numbers derived from a `count()`. To learn how to sort your table of contents before you output it, start by looking up `<xsl:sort>` at https://www.w3schools.com/xml/xsl_sort.asp or in Michael Kay. So far, if we've wanted to output, say, our table of contents in the order in which they occur in the document, we've used a self-closing empty element to select them with something like `<xsl:apply-templates select="$dickinsonColl//body"/>`. We've also said, though, that the self-closing empty element tag is informationally identical to writing the start and end tags separately with nothing between them, that is, `<xsl:apply-templates select="$dickinsonColl//body"></xsl:apply-templates>`. To cause the elements being processed to be sorted first, you need to use this alternative notation, with separate start and end tags, because you need to put the `<xsl:sort>` element between the start and end tags. If you use the first notation, the one with a single self-closing tag, there's no "between" in which to put the `<xsl:sort>` element. In other words, you want something like:

```
<xsl:apply-templates select="$dickinsonColl//body">
  <xsl:sort/>
</xsl:apply-templates/>
```

As written, the preceding will sort the `<body>` elements alphabetically by their text value. As you'll see at the sites mentioned above, though, it's also possible to use the `@select` attribute on `<xsl:sort>` to sort a set of items by properties other than alphabetic order of their textual content, which is what we will be doing in sorting on a `count()` of the `<rdg>` elements that we used to signal variant words and phrases in Dickinson's text.

Using `translate()` to fix the alphabetical sort order

If you sort the first lines alphabetically according to their textual value, there will be two errors. The first lines of Poem 6 and Poem 9, "'Tis so appalling—it exhilarates—" and "'Twas just this time, last year, I died.", will show up first because in the internal representation of characters in the computer, the single straight apostrophe is "alphabetically" earlier than all of the letters. We can fix this by using `translate()` to strip the apostrophe for sorting purposes, but not for rendering. That is, we can sort as if there were no apostrophe, while still printing the apostrophe when we render the line.

We can't easily translate away an apostrophe, though, because quotation marks have special meaning in XPath. For the purpose of this assignment, you can ignore these two missorted lines or let the apostrophe be sorted first. If you're feeling ambitious, though, read Michael Kay's answer at http://p2p.wrox.com/xslt/50152-how-do-you-translate-apostrophe.html and see whether you can apply it to fixing this problem.

Another Optional Challenge, for either of the table of contents or the body output, or both: You may have observed in your HTML output that some of our titles are inconsistently formatted. Some poem numbers have a period after them, and some only white space before the parenthetical information that summarizes each poem’s publication history. You might see if you can find a way to: a) output only the poem and its number in the part of the document where you reproduce the poems, and/or b) remove the rogue period from the output, using the `replace()` function, which takes three arguments (for "finding a needle in a haystack" and then changing or removing it): an XPath leading to a string you want to alter (your "haystack"), a regular expression for the "needle" you want to find and change, and whatever you wish to convert it to (including nothing, to delete it). Read about `replace()` in the Strings section of The XPath functions we use the most and learn about its syntax by looking it up in the index of the Michael Kay book.

Finishing touches

Some lists of first lines of poetry put quotation marks around the lines. We haven’t done that in our solution, but if you’d like to add it, you should use the HTML `<q>` ("quoted text") element, instead of outputting the raw quotation marks as plain text.

Oh, and did we mention CSS? Can you associate a CSS stylesheet to your output (write the CSS file link into your XSLT) to make it look more interesting than what you get by default in a web browser? See if you can find an interesting way to style the `<span>` elements surrounding the variants.