XML Technologies: Mandatory Exercise 3

XQuery, XSLT

November 18, 2014

Submission: November 25, 16:00

This is a **mandatory** exercise and the result will be part of your final mark. The solution must be uploaded to the OLAT folder "Übung 3" prior to **November 25, 16:00 - November 25, 15:59 at the very latest**. Late submissions will not be accepted.

Submit the following files in a zipped archive:

- ex01.xq, ex01.xml
- ex02.xsl, ex02.xml

Make sure the archive is named [firstname]_[lastname].zip (for example *mathias_mueller.zip*).

1 XQuery Revisited

XQuery has already been covered in the facultative Exercise 4. This question therefore assumes knowledge of FLWOR expressions, variable assignments and nested queries. Also, the query input XML will be the MEDLINE citations collection once again, you find it in the zip folder as `medsamp2014.xml`.

Designing an XML database like the MEDLINE citations involves a number of crucial decisions. One of them is deciding on a *point of view*. If you think of an XML document of a story that is being told, the point of view is embodied in a main character – through whose eyes the story unfolds. In the MEDLINE XML, the main character is the `MedlineCitation`.

This might be convenient in some situations, but in others it is not: for example, if you'd like to concentrate on authors, rather than on citations. Then, the organisation of this XML document is more of a hindrance. And that's why you will write an XQuery that tells the MEDLINE story from the point of view of authors.

Write an XQuery that finds all distinct last names in the MEDLINE document. Then, for each author name, list all the citations that belong to this author. Order the authors by name and the citations by year of journal publication. Your XML output should look like:

```
<!--PLEASE NOTE: This is a beautified result, for ease of reading.-->
<authors>
    <author name="Abbott">
        <citation title="Measurement........................." year="2002"/>
    </author>
    <author name="Abboud">
```

```
        <citation title="Effects ..........................." year="1999"/>
    </author>
    <!--Many more author elements-->
</authors>
```

Save your query as `ex01.xq` and the result as `ex01.xml`.

2 Project Euler and Recursive XSLT

All XSLT stylesheets we have seen so far are pretty short and straightforward. Yet, many real-world applications take this to quite another level of sophistication. Aspects of more advanced XSLT code are, for instance,

- organizing XLST code in separate modules that import or include one another, with an explicit hierarchy (modularisation)
- multi-step transformations with different template modes, in order to reiterate XML data that was processed already (where a template was applied already)
- recursive named templates that search arbitrarily deep structures or perform any other task an arbitrary number of times, depending on an input parameter

In this exercise, we will concern ourselves with the last of those aspects, writing a recursive template. More precisely, you are going to find a solution to the first problem of the Euler Project.

2.1 The Euler Problem 1

The Euler Project is best explained in its own words:

Project Euler is a series of challenging mathematical/computer programming problems that will require more than just mathematical insights to solve. Although mathematics will help you arrive at elegant and efficient methods, the use of a computer and programming skills will be required to solve most problems.

Also,

The intended audience include students for whom the basic curriculum is not feeding their hunger to learn (...)

and your tutor is confident you are that kind of student. The first of those Euler problems[1] reads:

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 1000.

2.2 A named template went missing

The Euler problem 1 was solved by more than 400.000 people and thus it is reasonable to assume that someone submitted an XSLT solution. Suppose that person gave you their XSLT solution – but somehow, the named template disappeared:

[1] The problem description is also available on the Euler project page: `https://projecteuler.net/problem=1`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <xsl:output indent="yes" omit-xml-declaration="yes"/>

    <xsl:template match="/">
        <xsl:variable name="k" select="1000"/>
        <xsl:variable name="multiples">
            <xsl:call-template name="find-multiples">
                <xsl:with-param name="k" select="$k"/>
            </xsl:call-template>
        </xsl:variable>

        <!--Output variable $multiples with, e.g.
            <xsl:value-of select="sum(
                                  for $x in tokenize($multiples, ' ') return number($x)
                                  )"/>
        -->
    </xsl:template>

    <xsl:template name="find-multiples">
       <!--Your code here-->
    </xsl:template>

</xsl:transform>
```

The incomplete stylesheet above is also available as `ex02.xsl` in the zip folder. The input to this stylesheet is irrelevant, but transform `euler1.xml` which you will also find in the zip folder.

Complete the stylesheet above by writing a named template that

- **integrates well with the scaffold that is already given (i.e. that does not require any change to the code outside of the named template)**
- **takes an input parameter called "k" where a number can be specified up to which multiples are found**
- **returns a sequence of all multiples – by returning an `xsl:sequence` for every multiple that is found, in a way that if you read out the value of `$multiples` like this:**

```
<xsl:variable name="multiples">
        <xsl:call-template name="find-multiples">
            <xsl:with-param name="k" select="$k"/>
        </xsl:call-template>
</xsl:variable>

<xsl:value-of select="$multiples"/>
```

the result would be:

```
0 3 5 6 9 10 12 15 18 20 21 24 25 27 30 33 35 36 39 40 42 45 48 50 (...)
```

Save the complete stylesheet as `e02.xsl` and the transformation result as `ex02.txt`