CSE344 fall 2013
Section 6 -- Practice with XML and XPath

This section, you will practice using XML and querying it with XPath which is a way to specify elements within an XML document using a notation reminiscent of the Unix file system.

We will use the Mondial database, which you are using in homework 5, due next week. Note that for homework 5, you will use XQuery instead of XPath, but every XPath expression is also an XQuery query.

Here is a simplified example of Mondial:

```
-------------------------------
<mondial>
	<continent id="europe">
		<name>Europe</name>
		<area>9562488</area>
	</continent>
	<country car_code="D" capital="cty-Germany-Berlin">
		<name>Germany</name>
		<population>83536115</population>
		<population_growth>0.67</population_growth>
		<infant_mortality>6</infant_mortality>
		<gdp_total>1452200</gdp_total>
		<indep_date>1871-01-18</indep_date>
		<government>federal republic</government>
		<encompassed continent="europe" percentage="100"/>
		<border country="F" length="451"/>
		<border country="A" length="784"/>
		<province id="prov-cid-cia-Germany-2" capital="cty-cid-cia-Germany-9">
			<name>Baden Wurttemberg</name>
			<city id="cty-cid-cia-Germany-9">
				<name>Stuttgart</name>
				<longitude>9.1</longitude>
				<latitude>48.7</latitude>
				<population year="95">588482</population>
			</city>
		</province>
	</country>
	<river id="river-Thjorsa" country="IS">
		<name>Thjorsa</name>
		<to watertype="sea" water="sea-Atlantic"/>
		<area>7530</area>
		<length>230</length>
		<source country="IS">
			<longitude>-18</longitude>
			<latitude>65</latitude>
		</source>
		<estuary country="IS">
			<longitude>-20.8</longitude>
			<latitude>63.9</latitude>
		</estuary>
	</river>
</mondial>
-------------------------------------------------
```

CSE344 fall 2013
Section 6 -- Practice with XML and XPath

We will use the Saxon XQuery/XPath processor. You can download it from here:
http://saxon.sourceforge.net
Select "Saxon Home Edition" (Saxon-HE). We'll use the Java version. Download and unzip the file to reveal the Java JAR file. Don't unzip by double-clicking on the Mac, because you might end up unzipping the JAR file too.
Then you can run Saxon on the command line. Cd to the directory with Saxon and say:

java -cp saxon9he.jar net.sf.saxon.Query YOUR_QUERY_FILE.xq

Unfortunately Saxon pushes all the output onto one line. An easy way to nicely display all the output is by redirecting it to a .xml file and then opening the XML file in a web browser:

```
... net.sf.saxon.Query YOUR_QUERY_FILE.xq > YOUR_QUERY_RESULT.xml

firefox YOUR_QUERY_RESULT.xml
```

IE and Firefox can display any XML file in a nicely readable form. Safari cannot, and only later versions of Chrome can. Note that no browser can display badly-formed XML; however, whenever an XPath query returns a list of items rather than just one item, Saxon will return badly formed XML in the output.
Practice XPath Queries for today:

Questions:

Q0. List the entire contents of Mondial.

```
doc("mondial.xml")/mondial
```

You can specify the file to read data from with the doc() XPath function. Then you can specify the root XML element with doc()/ELEMENT_NAME, just like a Unix path. (It's customary to omit the doc() on paper.)

Q1. Give a list of all the countries in XML.

```
doc("mondial.xml")//country
```

The double slash // before "country" tells XPath to find <country> elements at any point in the XML data tree below the point that precedes the double //. Because the // is preceded here by the doc(), this means to find <country> anywhere in Mondial.
Note that this will return badly formed XML. We can fix this by wrapping it in a dummy "answer" element:

```
<result>
{ doc("mondial.xml")//country }
</result>
```

Note that the curly braces are needed here to tell Saxon that you actually want to evaluate the XPath expression, instead of including it as a literal string.

Q2. Give a list of the countries that Germany borders.

```
<result>
{ doc("mondial.xml")//country[@car_code="D"]/border }
```

</result>

You can filter the elements returned by a boolean expression in square brackets []. Here, we ask for <country> elements whose "car_code" attribute (@car_code) is equal to "D", and then get the <border> elements who are the immediate children.

Q3. Give the names of all the countries with population at least 10 million.

```
<result>
{ doc("mondial.xml")//country[population/text() >= 10000000]/name }
</result>
```

To do the comparison, you need to obtain the character string within each <population> element. You do this by using the text() function of XPath as an immediate "child" of population/ . XPath will then coerce the string to a number automatically. Another way to write this query:

```
<result>
{ doc("mondial.xml")//country[population >= 10000000]/name }
</result>
```

(If an element has only text, its name can be used without having to specifically use the text() function of XPath.)

Q4. Find all cities located in countries that are partially or fully part of Europe. (The cities themselves don't have to be in Europe.)

```
<result>
{ doc("mondial.xml")//country[encompassed/@continent="europe"]//city }
</result>
```

Conditional expressions can have complex XPath expressions inside as well. Here we search for countries by matching an attribute of a <country>'s subelement.

Q5. Find the names of all rivers that start north of the equator (at a positive latitude).

```
<result>
{ doc("mondial.xml")//river[source/latitude > 0.0]/name }
</result>
```

Q6. Find the names of all rivers that start in Iceland.

```
<result>
{ doc("mondial.xml")//river[source/@country = (//country[name='Iceland']/@car_code)]/name }
</result>
```

Notice how we have nested one absolute XPath expression inside another - we compare the country attribute against the ID code of the country named Iceland.

Q7. Get the names of all countries in both Asia and Europe.

CSE344 fall 2013
Section 6 -- Practice with XML and XPath

```
<result>
{ doc("mondial.xml")//country[encompassed/@continent='europe' and
encompassed/@continent='asia']/name }
</result>
```

How does this work ?
In XPath, equality comparisons have implicit existential quantifiers. This means they return true if *one* of the items in the left-hand sequence matches *one* of the items in the right-hand sequence (either sequence can consist of just one item, such as 'europe' above). This is true of all comparison operators, actually. Hence, since there exists an <encompassed continent='europe' /> subelement, and another distinct <encompassed continent='asia' /> subelement, there can be a match.
This would not work:

```
<result>
{ doc("mondial.xml")//country/encompassed[@continent='europe' and @continent='asia']/name }
</result>
```

because here there really is only one item on each side of the equality test, there being only one "continent" attribute in an <encompassed>.
Alternative way:

```
<result>
{ doc("mondial.xml")//country[encompassed/@continent='europe'][encompassed/@continent='asia']/na
me }
</result>
```

XPath condition brackets stack from left to right.