Announcements

- Homework #3 assigned today (October 21)
  - Due on November 2
  - Beware of QuiP quirks
  - A fresher alternative: Saxon-B 8.1.1 (not supported)
  - Both QuiP and Saxon-B 8.1.1 can be installed on your own machine
- Graded Homework #2 available
  - Check your score on Blackboard

XSLT

- W3C recommendation
- XML-to-XML rule-based transformation language
- An XSLT program is an XML document itself
- Used most frequently as a stylesheet language

XSLT program

- An XSLT program is an XML document containing
  - Elements in the `<xsl:` namespace
  - Elements in user namespace
- The result of evaluating an XSLT program on an input XML document = the XSLT document where each `<xsl:` element has been replaced with the result of its evaluation
- Uses XPath as a sub-language

XSLT elements

- Element describing transformation rules
  - `<xsl:template>`
- Elements describing rule execution control
  - `<xsl:apply-templates>`
  - `<xsl:call-template>`
- Elements describing instructions
  - `<xsl:if>`, `<xsl:for-each>`, `<xsl:sort>`, etc.
- Elements generating output

XSLT example

- Find titles of books authored by "Abiteboul"

```xml
<?xml version="1.0"?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">
 <xsl:template match="book[author='Abiteboul']">
   <booktitle>
     <xsl:value-of select="title"/>
   </booktitle>
 </xsl:template>
</xsl:stylesheet>
```

- Not quite; we will see why later
<xsl:template>
  <xsl:template match="book[author='Abiteboul']">
    <booktitle>
      <xsl:value-of select="title"/>
    </booktitle>
  </xsl:template>
</xsl:template>

- <xsl:template match="match_expr"> is the basic XSLT construct describing a transformation rule
  - match_expr is an XPath-like expression specifying which nodes this rule applies to
- <xsl:value-of select="xpath_expr"/> evaluates xpath_expr within the context of the node matching the template, and converts the result node-set to a string
- <booktitle> and </booktitle> simply get copied to the output for each node match

Removing the extra output

- Add the following template:
  <xsl:template match="text()|@*"/>
- This template matches all text and attributes
- XPath features:
  - text() is a node test that matches any text node
  - @* matches any attribute
  - | means "or" in XPath
- Body of the rule is empty, so all text and attributes become empty string
  - This rule effectively filters out things not matched by the other rule

<xsl:attribute>

- Again, find titles of books authored by "Abiteboul"; but make the output look like <book title="booktitle"/>
  - <xsl:template match="book[author='Abiteboul']">
      <book>
        <xsl:attribute name="title">
          <xsl:value-of select="title"/>
        </xsl:attribute>
      </book>
  </xsl:template>
- A more general method:
  - <xsl:template match="book[author='Abiteboul']">
      <book>
        <xsl:attribute name="title">
          <xsl:value-of select="title"/>
        </xsl:attribute>
        <xsl:attribute name="author">
          <xsl:value-of select="author"/>
        </xsl:attribute>
      </book>
  </xsl:template>
  - <xsl:attribute name="attr">body</xsl:attribute>
  - <xsl:template>
      <xsl:attribute name="attr">body</xsl:attribute>
    </xsl:template>
- <xsl:attribute name="attr">body</xsl:attribute> adds an attributed named attr with value body to the parent element in the output

<xsl:copy-of>

- Another slightly different example: return (entire) books authored by “Abiteboul”
  - <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
      <xsl:stylesheet match="text()"/>
      <xsl:template match="book[author='Abiteboul']">
        <xsl:copy-of select="."/>
      </xsl:template>
    </xsl:stylesheet>
- <xsl:copy-of select="xpath_expr"/> copies the entire contents (including tag structures) of the node-set returned by xpath_expr to the output

Formatting XML into HTML

- Example templates to
  - Render a book title in italics in HTML
  - Render the authors as a comma-separated list
    - <xsl:template match="book/title">
      <i><xsl:value-of select="."/></i>
    </xsl:template>
    - <xsl:template match="book/author[1]">
      <xsl:value-of select="."/>
    </xsl:template>
    - <xsl:template match="book/author[position()>1]">
      <xsl:text>, </xsl:text><xsl:value-of select="."/>
    </xsl:template>
  - <xsl:template>
      <xsl:value-of select="."/>
    </xsl:template>
- <xsl:text> allows precise control of white space in output
Example: generate a table of contents
- Display books in an HTML unordered list
- For each book, first display its title, and then display its sections in an HTML ordered list
- For each section, first display its title, and then display its subsections in an HTML ordered list

```xml
<xsl:apply-templates />
```

Example continued
```xml
<xsl:template match="book">
  <li>
    <xsl:apply-templates select="title"/>
    <ol><xsl:apply-templates select="section"/></ol>
  </li>
</xsl:template>
```

One problem remains
- Even if a book or a section has no sections, we will still generate an empty `<ol>` element.

A fix using `<xsl:if>`: replace
```xml
<ol><xsl:apply-templates select="section"/></ol>
```
with
```xml
<xsl:if test="section">
  <ol><xsl:apply-templates select="section"/></ol>
</xsl:if>
```

The body of `<xsl:if test="xpath_cond">` is processed only if `xpath_cond` evaluates to true.

White space control
- White space is everywhere in XML.
  ```xml
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <!-- "\n" goes into a text node
    "\n\n" goes into another text node
  </book>
  ```
- Specify `<xsl:strip-space elements="*"/>` to remove text nodes (under any element) containing only white space.
- To strip leading and trailing white space and replace any sequence of white space characters by a single space, specify
  ```xml
  <xsl:template match="text()">
    <xsl:value-of select="normalize-space()"/>
  </xsl:template>
  ```

Named templates with parameters
- Define a generic template for rendering a list of things as a comma-separated list
  ```xml
  <xsl:template name="comma-separated-list">
    <xsl:param name="things-to-be-formatted"/>
    <xsl:for-each select="$things-to-be-formatted"/>
    <xsl:if test="position()>1">, </xsl:if>
    <xsl:value-of select="."/>
  </xsl:template>
  ```
- Cannot use match because we do not know in advance the things to render.
Calling templates & passing parameters

- Use the generic template
  
  ```xml
  <xsl:template match="book">
    <xsl:value-of select="title"/>
    <xsl:text>: </xsl:text>
    <xsl:call-template name="comma-separated-list">
      <xsl:with-param name="things-to-be-formatted" select="author"/>
    </xsl:call-template>
  </xsl:template>
  
  <xsl:with-param name="para_name" select="xpath_expr">
    evaluates xpath_expr and passes its result as the value of the parameter para_name
  </xsl:with-param>
  
  <xsl:call-template>
  invokes the named template without changing the context
  ```

XSLT summary

- Used often as a stylesheet language, but can be considered a query language too
  - Very expressive, with full recursion
    - Cannot be replaced by XQuery
  - Easily non-terminating, difficult to optimize
    - Cannot replace XQuery
- So many features, so little time! 😊

Review

- XML: tree (or graph)-structured data
- DTD: simple schema for XML
  - Well-formed XML: syntactically correct
  - Valid XML: well-formed and conforms to a DTD
- XPath: path expression language for XML
  - An XPath expression selects a list of nodes in an XML document
  - Used in other languages
- XQuery: SQL-like query language for XML
  - FLWOR expression, quantified expression, aggregation, etc.
- XSLT: stylesheet language for XML, in XML
  - Transforms input XML by applying template rules recursively on the structure of input XML