Announcements (Tue. Oct. 25)

- Homework #3 due in 1½ weeks
- Project milestone #2 due in 2½ weeks

XSLT

- XML-to-XML rule-based transformation language
  - Used most frequently as a stylesheet language
  - An XSLT program is an XML document itself
  - Current version is 2.0; W3C recommendation since January 2007

 Actually, output does not need to be in XML in general.
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
- Basic ideas
  - Templates specify how to transform matching input nodes
  - Structural recursion applies templates to input trees recursively
- 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 version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
  <xsl:template match="book[author='Abiteboul']">
    <booktitle>
      <xsl:value-of select="title"/>
    </booktitle>
  </xsl:template>
</xsl:stylesheet>
```

- Not quite; we will see why later
**Template in action**

<xsl:template match="book[author='Abiteboul']">
  <booktitle>
    <xsl:value-of select="title"/>
  </booktitle>
</xsl:template>

- **Example XML fragment**

  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
    <section>…</section>…
  </book>

    <title>A First Course in Databases</title>
    <author>Ullman</author>
    <author>Widom</author>
    <publisher>Prentice-Hall</publisher>
    <year>2002</year>
    <section>…</section>…
  </book>

- Template applies
  - `<booktitle>`
  - `<title>Foundations of Databases</title>`

- Template does not apply; default behavior is to process the node recursively and print out all text nodes
  - `<title>`
  - `<author>Ullman</author>`
  - `<author>Widom</author>`
  - `<publisher>Prentice-Hall</publisher>`
  - `<year>2002</year>`
  - `<section>…</section>…`

**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
Again, find titles of books authored by “Abiteboul,” but make the output look like

\[
\text{<book title="booktitle"/>}
\]

A more general method

\[
\text{xsl:template match="book[author='Abiteboul']">}
\text{<book title="title"/>}
\]

\[
\text{xsl:attribute name="attr">body</xsl:attribute>
\]

Another slightly different example: return (entire) books authored by "Abiteboul"

\[
\text{<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">}
\text{xsl:template match="text()|@*"/>}
\]

\[
\text{xsl:template match="book[author='Abiteboul']">}
\text{xsl:copy-of select=".”/>}
\]

Example templates to

- Render a book title in italics in HTML
- Render the authors as a comma-separated list

\[
\text{xsl:template match="book/title">}
\text{i><xsl:value-of select="normalize-space(.)"/></i>
\]

\[
\text{xsl:template match="book/author[1]">}
\text{xsl:value-of select="normalize-space(.)"/>}
\]

\[
\text{xsl:template match="book/author[position()>1]">}
\text{xsl:text>, </xsl:text>
\text{xsl:value-of select="normalize-space(.)"/>}
\]

\[
\text{xsl:template } \text{allows precise control of white space in output}
\]
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>
  <xsl:if test=""/>  
  <xsl:apply-templates select="section"/>  
</xsl:if>
</xsl:apply-templates>
```

Example continued
```xml
<xsl:if test="title">  
  <li>
    <xsl:apply-templates select="title"/>
    <ol>
      <xsl:apply-templates select="section"/>
    </ol>
  </li>
</xsl:if>
<xsl:apply-templates select="xpath_expr"/>
```

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

```xml
<xsl:if test=""/>
  <xsl:if test="section">  
    <ol>
      <xsl:apply-templates select="section"/>
    </ol>
  </xsl:if>
</xsl:if>
```

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

```xml
<xsl:output method="html" indent="yes"/>

- Specifies that output
  - Will be HTML
  - Will be indented to make reading easier
- Other possible method values include "text", "xml"
  - For XML output method, set
    omit-xml-declaration="yes"
    to suppress "<?xml ...?>" at the beginning of the output
```

White space control

- White space is everywhere in XML.
  ```xml
  <book ISBN="ISBN-10" price="80.00">
    <title>
      Foundations of Databases
    </title>
  </book>
  ```
  - "..." goes into a text node (assuming no DTD)
  - "..." goes into another text node
- 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>
  ```

```
xsl:for-each

- `<xsl:for-each select="xpath_expr">`
  ```xml
  <book>
    ...
  </book>
  </xsl:for-each>
  ```
  - Process body for each element in the node-set returned by xpath_expr
  - Processing context changes to the node being processed
- Another way to render authors as a comma-separated list
  ```xml
  <xsl:template match="book">
    ... ...
  </xsl:template>
  ```
```
Named templates with parameters

- Define a generic template for rendering a list of things as a comma-separated list
  - Cannot use `match` because we do not know in advance the things to render
  ```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="normalize-space(.)"/>
    </xsl:for-each>
  </xsl:template>
  ```

Calling templates & passing parameters

- Use the generic template
  ```xml
  <xsl:template match="book">
    <xsl:value-of select="normalize-space(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:call-template>` invokes the named template without changing the context

Other useful features

- `<xsl:text>&#10;</xsl:text>` inserts a newline in the output
- `<xsl:message>` for debugging
  - `<xsl:message terminate="yes">` exits the program
- `<xsl:variable>` defines a (constant) variable
- `<xsl:function>` defines a function
- `<xsl:key>` defines a key that can be used for lookups
XSLT summary

- Used often as a stylesheet language, but can be considered a query language too
  - Grouping in XSLT 2.0 (<xs1:for-each-group>)
  - Very expressive, with full recursion
    - Cannot be replaced by XQuery?
      - Well, XQuery supports user-defined functions, which can be recursive
  - 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
- XML Schema: a more sophisticated schema for XML
- 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