Announcements

- Homework #3 to be assigned next Monday (October 27)
- Course project milestone 2 due on November 12

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 processor

Input XML → XSLT program

Output XML

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
- 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"?>
<xs:s:stylesheet
 xmlns:xs="http://www.w3.org/1999/XSL/Transform"
 version="1.0">
 <xs:template match="book[author='Abiteboul']">
  <booktitle>
   <xs:valulce-of select="title"/>
  </booktitle>
 </xs:template>
</xs:s: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 match="match_expr">
    is the basic XSLT construct describing a transformation rule.
    <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.
  </xsl:template>

  <xsl:template match="text()|@*">
    This template matches all text and attributes.
    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:template>

Template in action

Example XML fragment

Template applies:
  <booktitle>Foundations of Databases</booktitle>
Template does not apply; default behavior is to process the node recursively and print out all text nodes.

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 `<book title="booktitle"/>

A more general method

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

Example templates to:
- Render a book title in italics in HTML
- Render the authors as a comma-separated list

Formatting XML into HTML
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:template match="title">
    <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="section">
    <li>
        <xsl:apply-templates select="title"/>
        <ol><xsl:apply-templates select="section"/></ol>
    </li>
</xsl:template>
```

(Continue on next slide)

Example continued

```xml
<xsl:template match="book">
    <li>
        <xsl:apply-templates select="title"/>
        <ol><xsl:apply-templates select="section"/></ol>
    </li>
</xsl:template>
<xsl:template match="bibliography">
    <html>
        <head><title>Bibliography</title></head>
        <body>
            <ul><xsl:apply-templates select="book"/></ul>
        </body>
    </html>
</xsl:template>
```

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

```xml
<xsl:if>
    <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

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

- White space is everywhere in XML.
- "Foundations of Databases" goes into a text node
- "Foundations of Databases" 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 `<xsl:template match="text()"> <xsl:value-of select="normalize-space()"/> </xsl:template>`

<xsl:for-each>
- `<xsl:for-each select="$xpath_expr"> body
  </xsl:for-each>`
  - Process body for each node 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
  `<xsl:template match="book">...
  <xsl:for-each select="author"> <xsl:if test="position()>1">, </xsl:if> <xsl:value-of select="."/> </xsl:for-each>
  </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
  `<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:for-each>
  </xsl:template>`
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: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! 😊