4. Unit: Transforming XML with XSLT

Exercise 4.1 (XML to HTML) Write an XSLT stylesheet that outputs the following country data for all countries with more than 1000000 inhabitants in an HTML table with rows of the form (property: value):

- country name
- car code
- capital's name
- number of inhabitants
- the names of all listed cities, inside a nested HTML table.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">
<xsl:output method="html" indent="yes"/>
<xsl:template match="mondial">
  <html> <body> <table>
          <xsl:apply-templates select="country[population > 1000000]"/>
      </table>  </body> </html>
</xsl:template>
<xsl:template match="country">
  <tr>
    <td colspan="2"><b>
       Country: <xsl:value-of select="name"/>
       </b></td>
  </tr>
  <tr>
    <td>car code:</td>
    <td><xsl:value-of select="@car_code"/></td>
  </tr>
  <tr>
    <td>capital:</td>
    <td><xsl:value-of select="id(@capital)/name"/></td>
  </tr>
  <tr>
    <td>population:</td>
    <td><xsl:value-of select="population"/></td>
  </tr>
  <tr>
    <td colspan="2">
      <table>
        <xsl:apply-templates select=".//city"/>
      </table>
    </td>
  </tr>
</xsl:template>
<xsl:template match="city">
  <tr><td>city:</td><td><xsl:value-of select="name"/></td></tr>
</xsl:template>
</xsl:stylesheet>
```

The following solution is more flexible and makes more use of XSL's template strategy: there are standard templates to output element, attribute, of reference properties. With this solution it is easy to list further properties in the output:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">
<xsl:output method="html" indent="yes"/>
<xsl:template match="mondial">
  <html> <body> <table>
          <xsl:apply-templates select="country[population > 1000000]"/>
      </table>  </body> </html>
</xsl:template>
<xsl:template match="country">
  <tr>
    <td colspan="2"><b>
       Country: <xsl:value-of select="name"/>
      </b></td>
  </tr>
  <xsl:apply-templates select="@car_code|@area|population"/>
  <xsl:apply-templates select="@capital" mode="ref"/>
  <tr>
    <td colspan="2">
      <table>
        <xsl:apply-templates select="city"/>
      </table>
    </td>
  </tr>
</xsl:template>
<xsl:template match="city">
  <tr><td>city:</td><td><xsl:value-of select="name"/></td></tr>
</xsl:template>
<xsl:template match="element()|attribute()">
  <tr>
    <td><xsl:value-of select="name()"/>:</td>
    <td><xsl:value-of select="."/></td>
  </tr>
</xsl:template>
<xsl:template match="attribute()" mode="ref">
  <tr>
    <td><xsl:value-of select="name()"/>:</td>
    <td><xsl:value-of select="id(.)/name"/></td>
  </tr>
</xsl:template>
</xsl:stylesheet>
```

Exercise 4.2 (XML to XML) Write an XSL stylesheet that, for every organization (optionally: that has been established before 1990), outputs its name, the names of all member countries, and the name of the city where its headquarter is located. The result should conform with the following DTD:

```
<!ELEMENT result (organization*)>
<!ELEMENT organization (country*, headq?)>
 <!ATTLIST organization name CDATA #REQUIRED>
<!ELEMENT country (#PCDATA)>
<!ELEMENT headq (#PCDATA)>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                version="2.0">
<xsl:output method="html" indent="yes"/>
<xsl:template match="mondial">
   <result>
    <xsl:apply-templates select="//organization[year-from-date(established) &lt; 1990]"/>
   </result>
</xsl:template>
<xsl:template match="organization">
  <organization name="{name}">
    <!-- optionally: use attribute constructor -->
    <xsl:apply-templates select="members/id(@country)"/>
    <xsl:if test="@headq">
      <headq> <xsl:value-of select="id(@headq)/name"/> </headq>
    </xsl:if>
  </organization>
</xsl:template>
<xsl:template match="country">
  <country> <xsl:value-of select="name"/> </country>
</xsl:template>
</xsl:stylesheet>
```

Exercise 4.3 (Arithmetic Terms)   Arithmetic terms over integer values and operators $+, -, *$ and *div* (integer division) can be represented by their syntax trees, with the syntax trees given in XML. A possible XML notation for syntax trees is given in the following example for the term

$$4 + ((7 - 2) \; div \; 2)$$

```
<term>
   <plus>
      <val>4</val>
      <div>
         <minus>
            <val>7</val>
            <val>2</val>
         </minus>
         <val>2</val>
      </div>
   </plus>
</term>
```

a) Write down the syntax tree for the term $((90 \; div \; (19 - (3 * 8))) + 3)$, using the XML notation from the above example.

b) Write a DTD for the given notation.

c) Write three XSLT stylesheets that take a syntax tree in the notation depicted above as input, and produce as output
- the term as text in *inorder* notation (outcome should be `(4+((7-2) div 2))` for the example),
- the term as text in *preorder* notation (outcome should be `+ 4 div - 7 2 2`), and
- the term as text in *postorder* notation (outcome should be `4 7 2 - 2 div +`).

Test the stylesheets using the term $((90 \ div \ (19 - (3 * 8))) + 3)$ as input.

d) Write an XSLT stylesheet that evaluates a syntax tree in the notation depicted above.

e) (Advanced) Extend the term notation, the DTD, and the solution of (4) to handle a `sum` operator that accepts an arbitrary number of operands.

Discussion of solution:

- discuss "algebra", consisting of a carrier set (here: numbers) and a set of operators (here: arithmetic ops) with arities. Illustrate generic markup of any algebra expression as (XML) tree.
- The stylesheets illustrate several possibilities how the problem can be solved.
- Stylesheet 1 (inorder): straightforward, using xsl:choose for distinguishing operators. Note that inorder is only well-defined for binary trees.
- Stylesheet 2 (preorder): each operator is represented by a separate template.
- Stylesheet 3 (all orders): each operator is represented by three templates. The "mode" selects which order is chosen.
  (provide the mode when calling the stylesheet, e.g., by `saxonXSL -im preorder arithm-tree-example.xml arithm-tree-inorder.xsl`)
- general: XSL for traversing a tree via template-calling and doing the right things: In the second variant, depending on the operator, the corresponding template is automatically selected. There is no explicit if/when etc.
- the evaluation stylesheet can be implemented by xsl:choose or by different templates. The first possibility is shorter, since the recursive call is always the same. But this is only possible since all operators are binary.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- XML instance for the term ((90 div (19 - (3*8)))+3) -->
<!DOCTYPE term SYSTEM "arithm-tree.dtd">
<term>
  <plus>
    <div>
      <val>90</val>
      <minus>
        <val>19</val>
        <mult>
          <val>3</val>
          <val>8</val>
        </mult>
      </minus>
    </div>
    <val>3</val>
  </plus>
</term>
```

```
<!-- DTD "arithm-tree.dtd" for the term syntax tree notation -->
<!-- with additional "sum" operator -->
<!ENTITY % operand "(plus|minus|mult|div|val|sum)">
<!ELEMENT term %operand; >
<!ELEMENT plus (%operand; ,%operand;)>
<!ELEMENT minus (%operand; ,%operand;)>
<!ELEMENT mult (%operand; ,%operand;)>
<!ELEMENT div (%operand; ,%operand;)>
<!ELEMENT val (#PCDATA)>
<!ELEMENT sum ((%operand;)+)>
```

```
<?xml version="1.0" encoding="utf-8"?>
<!-- call
  saxonXSL -im preorder arithm-tree-example.xml arithm-tree-inorder.xsl -->
<!-- xslt stylesheet for inorder output -->
<!-- straightforward design using choose -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0">
  <xsl:template match="term">
    <xsl:apply-templates select="*"/>
  </xsl:template>
  <xsl:template match="plus|minus|mult|div">
    <xsl:text>(</xsl:text>
    <xsl:apply-templates select="./*[1]"/>
    <xsl:choose>
      <xsl:when test="name()='plus'">
        <xsl:text> + </xsl:text>
      </xsl:when>
      <xsl:when test="name()='minus'">
        <xsl:text> - </xsl:text>
      </xsl:when>
      <xsl:when test="name()='mult'">
        <xsl:text> * </xsl:text>
      </xsl:when>
      <xsl:when test="name()='div'">
        <xsl:text> div </xsl:text>
      </xsl:when>
    </xsl:choose>
    <xsl:apply-templates select="./*[2]"/>
    <xsl:text>)</xsl:text>
  </xsl:template>
  <xsl:template match="val">
    <xsl:value-of select="./text()"/>
  </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  call saxonXSL arithm-tree-example.xml arithm-tree-preorder.xsl -->
<!-- xslt stylesheet for preorder output -->
<!-- each operator with an individual template -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0">
```

```
  <xsl:template match="term">
    <xsl:apply-templates select="*"/>
  </xsl:template>

  <xsl:template match="plus">
    <xsl:text> + </xsl:text>
    <xsl:apply-templates select="./*[1]"/>
    <xsl:text> </xsl:text>
    <xsl:apply-templates select="./*[2]"/>
  </xsl:template>
  <xsl:template match="minus">
    <xsl:text> - </xsl:text>
    <xsl:apply-templates select="./*[1]"/>
    <xsl:text> </xsl:text>
    <xsl:apply-templates select="./*[2]"/>
  </xsl:template>
  <xsl:template match="mult">
    <xsl:text> * </xsl:text>
    <xsl:apply-templates select="./*[1]"/>
    <xsl:text> </xsl:text>
    <xsl:apply-templates select="./*[2]"/>
  </xsl:template>
  <xsl:template match="div">
    <xsl:text> div </xsl:text>
    <xsl:apply-templates select="./*[1]"/>
    <xsl:text> </xsl:text>
    <xsl:apply-templates select="./*[2]"/>
  </xsl:template>

  <xsl:template match="val">
    <xsl:value-of select="./text()"/>
  </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0"?>
<!-- call
  saxonXSL -im postorder arithm-tree-example.xml arithm-tree-all-orders.xsl -->
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

<xsl:template match="term">
    <xsl:apply-templates select="*"/>
</xsl:template>

<xsl:template match="plus" mode="inorder">
  ( <xsl:apply-templates select="./*[1]"/>
    + <xsl:apply-templates select="./*[2]"/> )
</xsl:template>
<xsl:template match="plus" mode="postorder">
  <xsl:apply-templates mode="postorder" select="./*[1]"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates mode="postorder" select="./*[2]"/>
  <xsl:text> + </xsl:text>
</xsl:template>
```

```
<xsl:template match="minus" mode="inorder">
  ( <xsl:apply-templates select="./*[1]"/>
    - <xsl:apply-templates select="./*[2]"/> )
</xsl:template>
<xsl:template match="minus" mode="postorder">
  <xsl:apply-templates mode="postorder" select="./*[1]"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates mode="postorder" select="./*[2]"/>
  <xsl:text> - </xsl:text>
</xsl:template>
<xsl:template match="mult" mode="inorder">
  ( <xsl:apply-templates select="./*[1]"/>
    * <xsl:apply-templates select="./*[2]"/> )
</xsl:template>
<xsl:template match="mult" mode="postorder">
  <xsl:apply-templates mode="postorder" select="./*[1]"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates mode="postorder" select="./*[2]"/>
  <xsl:text> * </xsl:text>
</xsl:template>
<xsl:template match="div" mode="inorder">
  ( <xsl:apply-templates select="./*[1]"/>
    / <xsl:apply-templates select="./*[2]"/> )
</xsl:template>
<xsl:template match="div" mode="postorder">
  <xsl:apply-templates mode="postorder" select="./*[1]"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates mode="postorder" select="./*[2]"/>
  <xsl:text> / </xsl:text>
</xsl:template>

<xsl:template match="value">
  <xsl:value-of select="."/>
</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="utf-8"?>
<!-- call
  saxonXSL arithm-tree-example.xml arithm-tree-eval.xsl -->
<!-- xslt stylesheet for evaluating syntax tree terms -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0">
  <xsl:template match="term">
    <xsl:apply-templates select="*"/>
  </xsl:template>
  <xsl:template match="plus|minus|mult|div">
    <xsl:variable name="op1">
      <xsl:apply-templates select="./*[1]"/>
    </xsl:variable>
    <xsl:variable name="op2">
      <xsl:apply-templates select="./*[2]"/>
    </xsl:variable>
    <xsl:choose>
      <xsl:when test="name()='plus'">
        <xsl:value-of select="$op1 + $op2"/>
```

```
        </xsl:when>
        <xsl:when test="name()='minus'">
          <xsl:value-of select="$op1 - $op2"/>
        </xsl:when>
        <xsl:when test="name()='mult'">
          <xsl:value-of select="$op1 * $op2"/>
        </xsl:when>
        <xsl:when test="name()='div'">
          <xsl:value-of select="$op1 div $op2"/>
        </xsl:when>
      </xsl:choose>
  </xsl:template>
  <xsl:template match="val">
    <xsl:value-of select="./text()"/>
  </xsl:template>
</xsl:stylesheet>
```

Note (to myself):

• It is necessary to return the result as ‹val›*x*‹/val› and not just as number/text node since otherwise the results of the subterms will just be concatenated into a single text node.