4. Unit: XSLT

Exercise 4.1 (XML to HTML) Write an XSLT routine performing the following task: Map the following country data for each country to an HTML table:

- 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="1.0">
<xsl:template match="mondial">
  <html>
    <body>
      <table border="1">
        <xsl:apply-templates select="country[population/text() > 1000000]"/>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template match="country">
  <tr>
    <td>Country: <xsl:value-of select="name"/></td>
    <td>
      <table border="0">
      <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>
        <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>
```

Exercise 4.2 (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 $((91\ div\ (19 - (3 * 8))) + 3)$, using the XML notation from the above example.

b) Write a DTD for the given notation. Each term should be considered a single XML document instance.

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 $((91\ div\ (19 - (3 * 8))) + 3)$ as input.

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

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.
- Stylesheet 1: straightforward, using xsl:choose for distinguishing operators. Note that this works only as long as all operators have the same arity (consider the case that there is another, unary, "neg" operator).
- Stylesheet 2: each operator is represented by a separate template.
- 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.
- mention how stylesheets can be integrated in one using the XSL "mode" of templates.
- the evaluation stylesheet can be implemented in both ways.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- XML instance for the term ((91 div (19 - (3*8)))+3) -->
<!DOCTYPE term SYSTEM "arithmterm.dtd">
<term>
```

```
  <plus>
    <div>
      <val>91</val>
      <minus>
        <val>19</val>
        <mult>
          <val>3</val>
          <val>8</val>
        </mult>
      </minus>
    </div>
    <val>3</val>
  </plus>
</term>
```

```
<!-- DTD "term.dtd" for the term syntax tree notation -->
<!ELEMENT term (plus|minus|mult|div|val)>
<!ELEMENT plus ((plus|minus|mult|div|val),(plus|minus|mult|div|val))>
<!ELEMENT minus ((plus|minus|mult|div|val),(plus|minus|mult|div|val))>
<!ELEMENT mult ((plus|minus|mult|div|val),(plus|minus|mult|div|val))>
<!ELEMENT div ((plus|minus|mult|div|val),(plus|minus|mult|div|val))>
<!ELEMENT val (#PCDATA)>
```

```
<?xml version="1.0" encoding="utf-8"?>
<!-- xslt stylesheet for inorder output -->
<!-- straightforward design using choose -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <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"?>
<!-- xslt stylesheet for preorder output -->
<!-- each operator with an individual template -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <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" encoding="utf-8"?>
<!-- xslt stylesheet for evaluating syntax tree terms -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
  version="1.0">
  <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>
```

Exercise 4.3 (Recursion in Data)

(a) Write an XSLT stylesheet which maps the structures of the seas and rivers from Mondial in the following way: Every sea element must contain the name of the sea and a river element for each river flowing into that sea. Each river element, again, must recursively contain a river element for each river flowing into it, and so on:

```
<waters>
  <sea>
    <name>North Sea</name>
    <river>
      <name>Rhein</name>
      <length>...</length>
      <river>
        <name>Main</name>
        <length>...</length>
        <river>
          <name>Tauber</name>
          <length>...</length>
        </river>
        :
```

```
        </river>
        <river>
          <name>Neckar</name>
          <length>...</length>
          <river>
        </river>
        :
      </river>
    </sea>
  </waters>
```

(b) Write another stylesheet (that uses the output of the above one as input) which computes for each river that flows into a sea the total sum of the length of all rivers flowing (directly or transitively) into it, and output the results into a table.

(c) Write another stylesheet (that uses the original mondial.xml!) as input) which computes for each river that flows into a sea the total sum of the length of all rivers flowing (directly or transitively) into it, and output the results into a table.

Discussion of solution:

- (a): generate a hierarchical structure by recursive template calls.
- (b): use a recursive template call for computing a recursive function.
- solution to part (c): (XSLT source is contained in the repository but not published).
  - can be done by the usual XSLT commands (don't hack!)
  - have a look at the evaluation of terms of the previous exercise and use variables and template applications
  - analyze what is not possible an think how to replace it.

```
<!-- transformation of waters into recursive structure -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
<xsl:output method="html"/>
<xsl:template match="mondial">
  <waters>
    <xsl:apply-templates select="sea"/>
  </waters>
</xsl:template>
<xsl:template match="sea">
  <sea>
    <xsl:copy-of select="name"/>
    <xsl:apply-templates select="//river[id(to/@water)=current()]"/>
  </sea>
</xsl:template>
<xsl:template match="river">
  <river>
    <xsl:copy-of select="name"/>
    <xsl:copy-of select="length"/>
    <xsl:apply-templates select="//river[id(to/@water)=current()]"/>
  </river>
</xsl:template>
</xsl:stylesheet>
```

```
<!-- computing river network lengths from recursive structure -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<xsl:template match="*">
  <html>
    <table>
      <tr><td><b>Name</b></td><td><b>Length</b></td></tr>
      <xsl:apply-templates select="//sea/river"/>
    </table>
  </html>
</xsl:template>
<xsl:template match="river">
  <tr>
   <td>
    <xsl:value-of select="name"/>
   </td>
   <td>
    <xsl:choose>
      <xsl:when test="length">
        <xsl:value-of select="length + sum(.//river/length)"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>no length data available</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
   </td>
  </tr>
</xsl:template>
</xsl:stylesheet>
```