2. Unit: Transforming XML with XSLT

Exercise 2.1 (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.

Exercise 2.2 (Mondial - Deutschland-View)

Create a "Germany-View":
Use the web-resources

`http://www.geohive.com/cd/de.xml`
and
`http://www.geohive.com/cy/c_de.xml`

to create a view over Germany that satisfies the following DTD:

```
<!ELEMENT country (name,population,provinces,cities)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT population (#PCDATA)>
<!ELEMENT provinces (province*)>
<!ELEMENT cities (city*)>
<!ELEMENT province (name,area,population)>
<!ELEMENT area (#PCDATA)>
<!ELEMENT city (name,population)>

<!ATTLIST country capital IDREF #IMPLIED>
<!ATTLIST province capital IDREF #IMPLIED>
<!ATTLIST city id ID  #IMPLIED>
```

Hence, an instance of the view looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<country capital="...">
   <name>...</name>
   <population>...</population>
   <provinces>
      <province capital="...">
         <name>...</name>
         <area>...</area>
         <pop>...</pop>
      </province>
      ...
   </provinces>
   <cities>
        <city id="...">
           <name>...</name>
```

```
            <population>...</population>
        </city>
        ...
    </cities>
</country>
```

Each `city`-Element should get a unique ID with the `country` and `province` elements' `IDREF`-attributes referring to the adequate IDs of the addressed city.

Exercise 2.3 (Mondial - Transform Germany)

- Write an XSLT stylesheet performing the following task:
  write the "Germany" data from Mondial into a document that conforms to the DTD from Exercise 2.2.
- Invoke the stylesheet.
- Check if the resulting document is valid wrt. the DTD by using a validating XML parser, e.g. `xmllint`.

Exercise 2.4 (Stylesheet for Schedules)

Write an XSL stylesheet for transforming schedule data given in the following form into an html calendar table:

```
<?xml version="1.0" encoding="UTF-8"?>
<schedule name="Meine Termine">
        <month n="1">
            <day n="3">
                <date starttime="15:00">
                    <name>Institutsbesprechung</name>
                </date>
            </day>
        </month>
        <month n="2">
            <day n="2">
                <date starttime="18:00">
                    <name>Telefonat</name>
                </date>
            </day>
        </month>
    </year>
</schedule>
```

The resulting schedule should have about the following layout:

jan	feb	mar	apr	may	jun	jul	aug	sep	oct	nov	dec
1	1	1	1	1	1	1	1	1	1	1	1
2	2 18:00 - Telefonat	2	2	2	2	2	2	2	2	2	2
3 15:00 - Institutsbesprechung	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Exercise 2.5 (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.

Exercise 2.6 (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.

Exercise 2.7 (XSLT and Recursive Data Structures: Binary Search Trees)

a) Create a sample instance for XML Markup of *Binary Search Trees* (probably use the Abstract Datatype definition from the CS I lecture). Note: also an empty BST is a BST.

b) give a DTD-style description (note that since a BST can either be empty or not, the expressiveness of DTDs is not adequate). Optionally: give an XML Schema description.

c) write XSL stylesheets for (i) inserting elements into the tree, (ii) outputting the tree contents in inorder, (iii) outputting the tree in a graphical way (e.g., nested tables) in HTML, (iv) searching, and (iii) deleting elements in the tree.
(e.g., invoked by `saxonXSL -o mybsb.xml mybsb.xml bsb-insert.xsl insert=6`)