

EXAM IN "SEMI-STRUCTURED DATA" 184.705			24. 6. 2015
Study Code	Student Id	Family Name	First Name

Working time: 100 minutes.

Exercises have to be solved on this exam sheet; Additional slips of paper will not be graded.

First, please fill in your name, study code and student number. Please, prepare your student id.

**Exercise 1:**

(12)

Consider the following XML schema file **test.xsd**:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="E" type="EType"/>

  <xsd:complexType name="EType">
    <xsd:all>
      <xsd:element name="F" type="FType"/>
      <xsd:element name="G" type="GType"/>
    </xsd:all>
  </xsd:complexType>

  <xsd:simpleType name="FType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="([a-z][A-Z])+"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="GType">
    <xsd:choice>
      <xsd:element name="F" type="FType"/>
      <xsd:element name="G" type="GType" maxOccurs="2"/>
    </xsd:choice>
  </xsd:complexType>

</xsd:schema>
```

Consider additionally the following eight different XML files. All of the following files are well-formed. In this exercise you have to decide, which of the following are valid according to **test.xsd**.

1. <E><F>sToP</F><G></E> valid  invalid
2. <E><G><F>sToP</F></G></E> valid  invalid
3. <E><F>sToP</F><G><F>StOp</F></G></E> valid  invalid
4. <E><F>sToP</F><G><F>sToP</F></G></E> valid  invalid
5. <E><G><G><F>sToP</F></G><G><G><F>sToP</F></G></G></G></E> valid  invalid
6. <E><G><G><F>sToP</F></G><G><F>sToP</F></G></G><F>sToP</F></E> valid  invalid
7. <E><G><F>sToP</F><G><F>sToP</F></G></G><F>sToP</F></E> valid  invalid
8. <E><G><G><F>sToP</F></G></G><F>sToP</F></E> valid  invalid

(For every correct answer 1.5 points, for every incorrect answer -1.5 points, for every unanswered question 0 points, you can have at least 0 points on this exercise)

**Exercise 2:**

(15)

Decide which of the following statements is true or false.

1. A relational table can be represented as an XML document. true  false
2. XML is a presentation language like HTML. true  false
3. XML documents are stored as plain text. true  false
4. Every element in an XML document, including the root element, has exactly one parent. true  false
5. DTDs are written in XML syntax. true  false
6. In XML schema a complex element cannot have attributes. true  false
7. Tree-based parsers are faster than event-based parsers. true  false
8. In XPath “@” is the abbreviation for “attribute:.” true  false
9. Every XQuery expression can be written as an XPath expression. true  false
10. The “T” in XSLT stands for Template. true  false

(For every correct answer 1.5 points, **for every incorrect answer -1.5 points**, for every unanswered question 0 points, you can have at least 0 points on this exercise)

The following Exercises 3 – 7 are referring to the XML document `filesystem.xml`, which can be found on the last page of this exam.

**Exercise 3:**

(10)

Create a DTD document `filesystem.dtd`, s.t. the appended XML document `filesystem.xml` is valid. Consider the following specification:

- The root element of the document is called `filesystem`. It contains exactly one `cost` element and one `dir` element.
- The `cost` element contains a number, which stores the cost of traversing one directory in the filesystem.
- The `dir` element stores a directory of a filesystem. It has an optional `description` element, following an unbounded number of either `dir`, `file` or `link` elements. The `dir` element has a required `name` attribute.
- The `description` element contains a description of the directory. This is arbitrary text.
- The `file` element has a required attribute `size`, is identified by an `id` attribute and has an optional `type` attribute. The contents of the element is text (a filename).
- The `link` element has a required `ref` attribute, which refers to an `id` attribute of a `file` element. Its content is text (again a filename).
- If not specified, make reasonable assumptions on the types of the attributes and elements.

File `filesystem.dtd`:

```
<!ELEMENT filesystem (cost,dir)>
<!ELEMENT dir (description?,(dir|file|link)*)>
<!ATTLIST dir name CDATA #REQUIRED>
<!ELEMENT file (#PCDATA)>
<!ATTLIST file id ID #REQUIRED>
<!ATTLIST file size CDATA #REQUIRED>
<!ATTLIST file type CDATA #IMPLIED>
<!ELEMENT link (#PCDATA)>
<!ATTLIST link ref IDREF #REQUIRED>
<!ELEMENT description (#PCDATA)>
<!ELEMENT cost (#PCDATA)>
```

**Exercise 4:**

(10)

Consider the following XPath expressions and evaluate them over the appended XML document **filesystem.xml**.

- If the expression selects several nodes, separate the output with whitespaces.
- If the XPath expression selects no nodes, write “No output!”.

Write the output of the following expressions:

```
//file[text() = 'jeopardy']/@id
```

```
f g c e
```

```
//file[@id = //link/@ref]/@size
```

```
2000 2500
```

```
//dir[not(link)]/@name
```

```
src ssd
```

```
sum(//file[@size>4000]/@size)
```

```
9100
```

```
//dir[@name = 'src']/descendant-or-self::file/@id
```

```
a b
```

**Exercise 5:**

(8)

Consider the following XQuery statement **filesystem.xq**:

```
for $b in //file
let $p := $b/@type
where $b/@size < 3000
order by number($b/@size) descending
return <file> {concat(concat($b,'.'),$p)} - {string($b/@size)} </file>
```

Write the output of **filesystem.xq** evaluated over **filesystem.xml** here.  
Whitespaces don't have to be formatted correctly.

```
<file>jeopardy.xml - 2500</file>
<file>jeopardy.xml - 2000</file>
<file>query-out.xml - 1500</file>
<file>jeopardy.xsd - 1000</file>
<file>text. - 10</file>
```

**Exercise 6:**

(10)

Create an XSLT stylesheet **filesystem.xml**, which, applied to **filesystem.xml**, outputs an XML document having some filesystem statistics as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<stats>
  <ssd-ss15>
    <files>8</files>
    <size>19160</size>
    <src>
      <files>2</files>
      <size>7150</size>
      <ssd>
        <files>2</files>
        <size>7150</size>
      </ssd>
    </src>
    <output>
      <files>3</files>
      <size>9000</size>
    </output>
  </ssd-ss15>
</stats>
```

This means:

- The root element is **stats**
- For each directory with name **D**, create a **D** element. In this **D** element, the number of files stored in the subtree of the directory with name **D** is output in the **files** element, and the total size of these files in the **size** element.
- The above applies recursively to all subdirectories.

Write the stylesheet here **filesystem.xml**.

File **pruefung.xml**:

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>

  <xsl:template match="filesystem">
    <stats><xsl:apply-templates select="dir" /></stats>
  </xsl:template>

  <xsl:template match="dir">
    <xsl:element name="{@name}">
      <files><xsl:value-of select="count(descendant::file)" /></files>
      <size><xsl:value-of select="sum(descendant::file/@size)" /></size>
      <xsl:apply-templates select="dir" />
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

Complete the following SAX handler, which, applied to the XML document `filesystem.xml`, should output for a given *filename* and *type* the costs of traversing to this file. The costs are calculated as follows: The basic traversal cost is stored in the `cost` element. For each directory traversal the total costs are increased by this basic cost. E.g. given the filename “jeopardy” and the type “xml” the handler should output the following:

Cost for file <jeopardy.xml>: 20

Cost for file <jeopardy.xml>: 40

There are two files with the name “jeopardy” and type “xml”. One can be accessed with costs of “20” (in the directory “ssd-ss15”), the other with costs of “40” (in the directory “ssd-ss15/output”).

The format of the output is not important. The *filename* and the *type* are stored in the corresponding variables. The variable *eleText* contains the text of the last text element.

```
public class CountCost extends DefaultHandler {

    String eleText;
    String filename;
    String type;

    public void characters(char[] text, int start, int length) throws SAXException {
        eleText = new String(text, start, length); }

    public CountCost(String filename, String type) {
        this.filename = filename; this.type = type; }

    //Additional variables
    String seenType;
    int cost;
    int totalCost = 0;

    public void startElement(String namespaceURI, String localName, String qName, Attributes atts)
        throws SAXException {
        if ("dir".equals(localName)) totalCost += cost;
        if ("file".equals(localName)) seenType = atts.getValue("type");
    }

    public void endElement(String namespaceURI, String localName, String qName)
        throws SAXException {
        if ("cost".equals(localName)) cost = Integer.parseInt(eleText);
        if ("dir".equals(localName)) totalCost -= cost;
        if ("file".equals(localName)) {
            if (eleText.equals(filename) && seenType.equals(type))
                System.out.println("Cost for file <" + filename + "." + type + ">: " + totalCost);
        }
    }
}
```



Total points: 75



You can remove this sheet!

File filesystem.xml:

```
<filesystem>
  <cost>20</cost>
  <dir name="ssd-ss15">
    <description><![CDATA[ This directory stores <b>Jeopardy</b> files. ]]></description>
    <dir name="src">
      <description><![CDATA[ Java files belong here. ]]></description>
      <dir name="ssd">
        <file size="3050" type="java" id="a">SSD</file>
        <file size="4100" type="java" id="b">JeopardyMoveHandler</file>
      </dir>
    </dir>
    <file size="2000" type="xml" id="f">jeopardy</file>
    <file size="1000" type="xsd" id="g">jeopardy</file>
    <link ref="c">jeopardy-out.xml</link>
    <file size="10" id="h">text</file>
    <dir name="output">
      <link ref="f">jeopardy.xml</link>
      <file size="2500" type="xml" id="c">jeopardy</file>
      <file size="1500" type="xml" id="d">query-out</file>
      <file size="5000" type="html" id="e">jeopardy</file>
    </dir>
  </dir>
</filesystem>
```