EXAM IN “SEMI-STRUCTURED DATA” **184.705**			28. 10. 2016
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**:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="A">
    <xsd:complexType mixed="true">
      <xsd:choice minOccurs="2" maxOccurs="3">
        <xsd:sequence>
          <xsd:element name="B" minOccurs="0" type="xsd:int"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element name="C" maxOccurs="3" type="myint"/>
          <xsd:element name="D" minOccurs="0" type="xsd:int"/>
        </xsd:sequence>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="myint">
    <xsd:restriction base="xsd:int">
      <xsd:minExclusive value="1"/>
      <xsd:maxInclusive value="6"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

Furthermore, consider the eight different XML files, which are listed below.

You may assume that each of the following XML files is well-formed. The point is to determine the validity according to **test.xsd**.

Check which of the following XML files are valid according to **test.xsd**.

1. `<A><B/><B/></A>` valid ○ [✓ invalid]
2. `<A><C>1</C><D>2</D></A>` valid ○ [✓ invalid]
3. `<A>abc<C>6</C>def<C>2</C><B>0</B></A>` [✓ valid] invalid ○
4. `<A><C>6</C><C>2</C><C>5</C><C>4</C>abc<D>3</D></A>` [✓ valid] invalid ○
5. `<A><B>10</B></A>` [✓ valid] invalid ○
6. `<A><B>1</B><D>1</D>abc<C>2</C></A>` valid ○ [✓ invalid]
7. `<A><B>abc</B><C>2</C><D>2</D></A>` valid ○ [✓ invalid]
8. `<A><B>2</B><C>2</C></A>` [✓ 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 are true or false.

1. To check whether an XML document is well-formed, a DTD or an XML-Schema is required. true ◯ [✓ false ⊗]
2. The "X" in XML stands for eXecutable. true ◯ [✓ false ⊗]
3. In XML-Schema attributes have an arbitrary `xsd:simpleType` as data type. [✓ true ⊗] false ◯
4. DTD declaration `(A+|B+|C+)` is not equivalent to `(A|B|C)+`. [✓ true ⊗] false ◯
5. XML-Schema documents are not XML documents. true ◯ [✓ false ⊗]
6. XSLT is a W3C recommendation. [✓ true ⊗] false ◯
7. XPath is used to modify XML documents. true ◯ [✓ false ⊗]
8. Event-based parsers use a constant amount of memory. [✓ true ⊗] false ◯
9. XQuery is a schema language. true ◯ [✓ false ⊗]
10. The XPath test "`./../* eq .`" returns in every element (except the root element) *true*. 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 textref.xml, which can be found on the last page of this exam.

Exercise 3: (12)

Complete the DTD **textref.dtd**, so that XML documents structured like **textref.xml** (see attachment) are valid according to this DTD. Consider the following points when creating the DTD:

- `document` contains exactly one `main` element, at most one `comment` element and any number of `part` elements. `part` elements have to exist only at the end. The `comment` element is optional. If existing, it must be positioned right before or after the `main` element.
- The `main` element has mixed content, sub-elements are `author` and `ref`.
- `part` elements have mixed content with sub-elements `ref` and
  - a required attribute `id` with a unique attribute value.
- `ref` elements don't have any content, merely
  - a required attribute `to` which refers to the `id` of a `part` element.
- `author` elements and the `comment` element have simple text content.

File **textref.dtd**:

```
<!ELEMENT document (((comment, main) | (main, comment?)),  part*)>
<!ELEMENT main (#PCDATA | author | ref)*>
<!ELEMENT part (#PCDATA | ref)*>
<!ATTLIST part id ID #REQUIRED>
<!ELEMENT ref EMPTY>
<!ATTLIST ref to IDREF #REQUIRED>
<!ELEMENT author (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
```

Exercise 4: (10)

Consider the following XPath queries applied to the document **textref.xml** (see attachement).

- If a node set is selected as the result, write as output the value of the `id` attribute.
- If the given XPath expression selects the empty node set, write as output “empty output”
- If a number is selected as the result (count), write as output this number.

Consider the following example:

```
//part
```

```
p1   p2   p3   p4
```

Now give the outputs of the respective XPath queries:

```
count(//main//ref[@to='p4'])
```

```
2
```

```
//part[*]
```

```
p2   p3
```

```
/document/*[4]
```

```
p2
```

```
//part[@id = //ref/@to]
```

```
p1   p2   p3   p4
```

```
//part[@id = ref/@to]
```

```
p3
```

Exercise 5: (9)

Create an XSLT-Stylesheet **textref.xsl**, which returns the following result applied to documents of the form **textref.xml**:

- The output shall be the content of the `main` element.
- Instead of the `ref` elements, the content of the respective `part` elements shall be returned. This holds for the references directly contained in `main`, as well as any references contained in `part` elements.

The output shall be pure text (i.e. no XML markup shall be contained in it).

Consider the following output that your XSLT-Stylesheet **textref.xl** shall return applied to **textref.xml**:

```
    My text is written by me. With this complex text the nobel prize is not to far.
```

Now complete the XSLT-Stylesheet **textref.xsl**. Control structures like `xsl:for-each` are allowed, but not needed for the solution (sufficient for solving this exercise are few templates with little content). You do not need to consider whitespace issues.

File **textref.xsl**:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>

<xsl:template match="/">
    <xsl:apply-templates select="//main/node()"/>
</xsl:template>

<xsl:template match="ref">
    <xsl:apply-templates select="//part[@id=current()/@to]"/>
</xsl:template>

</xsl:stylesheet>
```

Exercise 6: (9)

Consider the following XQuery expression **xquery.xq**:

```
<statistics>{

  for $a in doc('textref.xml')/document/*
  let $count := count($a//*)
  return
     <c cnt="{$count}"><t>{$a/@id}</t></c>

}</statistics>
```

Now give the output of **xquery.xq** applied to **textref.xml**.

You do not need to consider whitespace issues.

```
<statistics>
   <c cnt="0"><t/></c>
   <c cnt="5"><t/></c>
   <c cnt="0"><t>p1</t></c>
   <c cnt="1"><t>p2</t></c>
   <c cnt="1"><t>p3</t></c>
   <c cnt="0"><t>p4</t></c>
</statistics>
```

Exercise 7: (8)

Complete the method `editRef`, which applied to a DOM element `ref` (a `ref` element of a document of the form **textref.xml**) makes the following changes: The `ref` element of the form

```
<ref to='px'/>
```

is replaced by an element of the form

```
<include part='px'/>
```

The method `editRef` furthermore has access to the parent of the `ref` element. This parent is necessary to perform the replacement.

You do not need to be concerned with error handling in this task.

```
public static void editRef(Element ref, Element parent) {
  String to = ref.getAttribute("to");
  Element include = ref.getOwnerDocument().createElement("include");
  include.setAttribute("part", to);

  parent.replaceChild(include, ref);
}
```

Total points: 75

You may separate this page!

File **textref.xml**:

```
<document>

    <comment>Text comment</comment>
    <main>
       My <ref to="p1"/> <ref to="p4"/> written
       by <author>me</author>. With
       this <ref to="p2"/> the
       nobel prize <ref to="p4"/> not
       to far.
    </main>

    <part id="p1">text</part>
    <part id="p2">complex <ref to="p1"/></part>
    <part id="p3">far <ref to="p3"/></part>
    <part id="p4">is</part>

</document>
```