EXAM IN "SEMI-STRUCTURED DATA" **184.705**			30. 11. 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 DTD schema file **test.dtd**:

```
<!ELEMENT A ((A|B), C?, B)>

<!ELEMENT B (#PCDATA|A|C)*>

<!ELEMENT C EMPTY>

<!ATTLIST A key ID #REQUIRED>

<!ATTLIST C choice (a|b|c|d) #IMPLIED>
```

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.dtd**.

1. `<A/>` valid ◯ invalid ⊗
2. `<A key="a"><B><C/></B><B>text</B></A>` valid ⊗ invalid ◯
3. `<A key="a"><B/><B><A key="b"><B/><B/><B/></A></B></A>` valid ◯ invalid ⊗
4. `<A key="a"><B/><C choice="e"/><B/></A>` valid ◯ invalid ⊗
5. `<A key="a"><B><C choice="a"/></B><C choice="b"/><C choice="c"/></A>` valid ◯ invalid ⊗
6. `<A key="a"><A key="b"><B/><B/></A><C/><B/></A>` valid ⊗ invalid ◯
7. `<A key="a"><B/><B><A key="a"><B/><B/></A></B></A>` valid ◯ invalid ⊗
8. `<A key="a"><B/><B/></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 is true or false.

1. Semi-structured data is a special case of structured data. true ◯ false ⊗
2. XML is a markup language. true ⊗ false ◯
3. XML documents are plain text. true ⊗ false ◯
4. XML documents can be executable files. true ◯ false ⊗
5. Validating errors can be ignored. true ⊗ false ◯
6. The "T" in DTD stands for transformation. true ◯ false ⊗
7. DTDs are XML documents. true ◯ false ⊗
8. XML Schemas are more powerful than DTDs. true ⊗ false ◯
9. XPath is a query language. true ⊗ false ◯
10. XPath is more powerful than XSLT. 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 – 6 are referring to the XML document products.xml, Exercises 6 and 7 are referring to the XML document products-xsl.xml. Both can be found on the last page of this exam.

Exercise 3: (14)

Create an XML Schema document **products.xsd** such that the **products.xml** document is valid. Consider the following specification:

- The root element of the document is called `products`. It contains zero or an unbounded number of `product` elements.
- The `product` element has an attribute `id` and contains the following elements in the given order: first, exactly one `name` element; second, one `value` element; and then, zero or an unbounded number of `product` or `productref` elements in arbitrary order.
- The `productref` element is an empty element and has an attribute `ref`.
- The `name` element contains a string.
- The `value` element contains an integer greater or equal than 0.
- Add a key `productKeys` identifying all products by their `id` attribute.
- Add a key reference `productRefs`. The `ref` attribute of the `productref` elements refer to the `productKeys`.

File **products.xsd**:

```
<!-- More space on the following page! -->

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

   <xsd:element name="products" type="products">
       <xsd:key name="productKeys">
           <xsd:selector xpath=".//product"/>
           <xsd:field xpath="@id"/>
       </xsd:key>

       <xsd:keyref name="productRefs" refer="productKeys">
           <xsd:selector xpath=".//productref"/>
           <xsd:field xpath="@ref"/>
       </xsd:keyref>
   </xsd:element>

   <xsd:complexType name="products">
       <xsd:sequence>
           <xsd:element name="product" type="product" minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="product">
       <xsd:sequence>
           <xsd:element name="name" type="xsd:string" />
           <xsd:element name="value" type="xsd:nonNegativeInteger" />
           <xsd:choice minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="product" type="product"   />
              <xsd:element name="productref" type="productref" />
           </xsd:choice>
       </xsd:sequence>
       <xsd:attribute name="id" type="xsd:integer" use="required"/>
   </xsd:complexType>
```

```
    <xsd:complexType name="productref">
        <xsd:attribute name="ref" type="xsd:integer" use="required"/>
    </xsd:complexType>

</xsd:schema>
```

Exercise 4: (8)

Consider the following XPath expressions and evaluate them over the **product.xml** document.

- 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:

```
count(/*/product)
```

2

```
//product[sum(.//value)<1000]/@id
```

2 5 4

```
sum(//product[@id = //productref/@ref]/value)
```

1000

```
//product[@id = 5]/preceding::product/@id
```

2

Exercise 5: (6)

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

```
for $ref in //productref
let $p := //product[@id = $ref/@ref]
order by $p/name
return $p
```

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

```
<product id="3">
    <name>Product C</name>
    <value>400</value>
    <product id="5">
        <name>Product E</name>
        <value>600</value>
    </product>
</product>
<product id="5">
    <name>Product E</name>
    <value>600</value>
</product>
```

Exercise 6: (10)

Create an XSLT stylesheet **products.xsl** that, after applied to **products.xml**, outputs the XML document **products-xsl.xml**. The idea is to generate a document that substitutes every `productref` element by the referenced `product` element. This means:

- The root element is `products`
- For each `product` element: create a `product` element and copy the name and value of this product. Apply templates to all `product` and `productref` child elements.
- For each `productref` element: Use `<xsl:variable name="ref" select="@ref" />` to store the value of the attribute `ref` into a variable `$ref`. Use this variable in an XPath expression to apply the template of the product with `id` equal to `$ref`.

Write the stylesheet here **products.xsl**.

File **pruefung.xsl**:

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

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

   <xsl:template match="product">
      <product>
        <name><xsl:value-of select="name" /></name>
        <value><xsl:value-of select="value" /></value>
        <xsl:apply-templates select="product" />
        <xsl:apply-templates select="productref" />
      </product>
   </xsl:template>

   <xsl:template match="productref">
      <xsl:variable name="ref" select="@ref" />
      <xsl:apply-templates select="//product[@id = $ref]" />
   </xsl:template>



</xsl:stylesheet>
```

Exercise 7: (10)

Complete the following SAX handler that, after applied to the **product-xsl.xml** document, outputs the name of the `product` elements that are child elements of the root and the sum of the `value` elements of all of their descendants. The output should be as follows:

```
Total value of Product A: 1900
Total value of Product D: 1300
```

For example, "Product A" has as descendants "Product B", "Product E", "Product C" and again "Product E". The total value of all these products is *100 + 200 + 600 + 400 + 600 = 1900*.

The format of the output is not important.

```
public class TopProducts extends DefaultHandler {

    String eleText;
    private int level = 0;
    private int value = 0;

    @Override
    /**
     * SAX calls this method to pass in character data
     */
    public void characters(char[] text, int start, int length)
                  throws SAXException {
       eleText = new String(text, start, length);
    }


    public void startElement(String namespaceURI, String localName, String qName, Attributes atts)
                  throws SAXException {
        if ("product".equals(localName)) {
           level++;
        }
    }

    public void endElement(String namespaceURI, String localName, String qName)
                  throws SAXException {
        if ("name".equals(localName)) {
           if (level == 1) System.out.print("Total value of " + eleText + ": ");
        }
        if ("value".equals(localName)) {
            value += Integer.parseInt(eleText);
        }
        if ("product".equals(localName)) {
            level--;
            if (level == 0)  {
                System.out.println(value);
                value = 0;
            }
        }
    }
}
```

Total points: 75

You can remove this sheet!

File **products.xml**:

```
<products>
   <product id="1">
      <name>Product A</name>
      <value>100</value>
      <product id="2">
         <name>Product B</name>
         <value>200</value>
         <productref ref="5" />
      </product>
      <product id="3">
         <name>Product C</name>
         <value>400</value>
         <product id="5">
            <name>Product E</name>
            <value>600</value>
         </product>
      </product>
   </product>
   <product id="4">
      <name>Product D</name>
      <value>300</value>
      <productref ref="3" />
   </product>
</products>
```

File **products-xsl.xml**:

```
<products>
    <product>
        <name>Product A</name>
        <value>100</value>
        <product>
            <name>Product B</name>
            <value>200</value>
            <product>
                <name>Product E</name>
                <value>600</value>
            </product>
        </product>
        <product>
            <name>Product C</name>
            <value>400</value>
            <product>
                <name>Product E</name>
                <value>600</value>
            </product>
        </product>
    </product>
    <product>
        <name>Product D</name>
        <value>300</value>
        <product>
            <name>Product C</name>
            <value>400</value>
            <product>
                <name>Product E</name>
                <value>600</value>
            </product>
        </product>
    </product>
</products>
```