Gruppe A	PRÜFUNG AUS "SEMISTRUKTURIERTE DATEN" **181.135**		21. 10. 2010
Kennnr.	Matrikelnr.	Familienname	Vorname

Arbeitszeit: 120 Minuten. Aufgaben sind auf den Angabeblättern zu lösen; Zusatzblätter werden nicht gewertet.
Bitte tragen Sie Namen, Studienkennzahl und Matrikelnummer ein. Halten Sie Ihren Studentenausweis bereit.

Aufgabe 1: (9)

Betrachten Sie die folgende XML-Schema Datei **test.xsd**:

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

  <xsd:element name="A">
    <xsd:complexType mixed="true">
      <xsd:choice minOccurs="1" maxOccurs="2">
        <xsd:sequence>
          <xsd:element name="B" minOccurs="0" type="xsd:string"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element name="C" maxOccurs="unbounded" type="xsd:string"/>
        </xsd:sequence>
     </xsd:choice>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Betrachten Sie weiters die acht verschiedenen XML-Dateien, die unten angeführt sind.

Sie können davon ausgehen, dass alle folgenden XML-Dateien wohlgeformt sind. Es geht also lediglich darum, ihre Gültigkeit bezüglich **test.xsd** zu entscheiden.

Kreuzen Sie an, welche der folgenden XML-Dateien gültig bezüglich **test.xsd** sind.

1. `<A><B><C>abc</C></B></A>` gültig ◯ ungültig ◯
2. `<A><B>abc</B></A>` gültig ◯ ungültig ◯
3. `<A>abc</A>` gültig ◯ ungültig ◯
4. `<A><B>abc</B><B>def</B><B>ghj</B></A>` gültig ◯ ungültig ◯
5. `<A><C>abc</C><C>def</C><C>ghj</C></A>` gültig ◯ ungültig ◯
6. `<A></A>` gültig ◯ ungültig ◯

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrekter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Aufgabe 2: (6)

Betrachten Sie die folgende XML- Datei **ns.xml**:

```
<A xmlns:ns="URI_A">
  <ns:B xmlns="URI_B">
    <D>abc</D>
  </ns:B>
  <C xmlns="URI_A">
    <ns:E xmlns:ns="URI_B">
  <C>
</A>
```

Kreuzen Sie an, ob die folgenden Aussagen für die Datei **ns.xml** wahr oder falsch sind.

1. Elemente `A` und `B` liegen im selben Namespace. wahr ◯ falsch ◯
2. Elemente `A` und `C` liegen im selben Namespace. wahr ◯ falsch ◯
3. Elemente `B` und `C` liegen im selben Namespace. wahr ◯ falsch ◯
4. Elemente `D` und `E` liegen im selben Namespace. wahr ◯ falsch ◯

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrekter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Aufgabe 3: (12)

Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

1. CDATA-Sections werden von XML-Parsern ignoriert. wahr ◯ falsch ◯
2. Die DTD Elementdeklaration `(A+|B+|C+)` ist gleichwertig mit `(A|B|C)+`. wahr ◯ falsch ◯
3. Mit XSLT ist es möglich, XML Dokumente in beliebige andere Dokumente (zB Text, HTML etc) zu transformieren. wahr ◯ falsch ◯
4. Ein XSLT-Stylesheet muss ein wohlgeformtes XML-Dokument sein. wahr ◯ falsch ◯
5. SAX verwendet die DOM API um ein XML Dokument zu lesen. wahr ◯ falsch ◯
6. Der Speicherbedarf eines DOM-Parsers ist unabhängig von der Größe des geparsten XML-Files. wahr ◯ falsch ◯
7. DOM erlaubt wahlfreien Zugriff auf das gesamte XML Dokument. wahr ◯ falsch ◯
8. Bei DOM sind die Attributwerte direkt im Elementknoten gespeichert. wahr ◯ falsch ◯

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrekter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Die folgenden Aufgaben 4 – 7 beziehen sich auf das XML-Dokument checklist.xml, das Sie auf der letzten Seite dieser Prüfungsangabe finden.

Aufgabe 4: (12)

Vervollständigen Sie die DTD **checklist.dtd**, sodass XML-Dokumente in der Gestalt von **checklist.xml** (siehe Anhang) bezüglich dieser DTD gültig sind. Berücksichtigen Sie beim Erstellen der DTD folgende Punkte:

- `checklist` enthält genau ein Kindelement `list`
- Ein `list` Element hat
  - optional ein Attribut `titel` und
  - ein Attribut `id` das nicht fehlen darf und einen eindeutigen Attributwert beinhalten soll.
- Ein `list` Element beinhaltet beliebig viele `li` Elemente, jedoch zumindest ein `li` Element. `li` Elemente können, müssen aber nicht, von `hline` Elementen getrennt werden. Zwei `hline` Elemente dürfen allerdings nicht unmittelbar hintereinander folgen.
- Der Inhalt eines `li` Elements soll gemischt sein, es dürfen als Subelemente `b`, `i` und erneut `list` Elemente auftreten.
- `b` und `i` Elemente sollen Textinhalt haben.
- `hline` soll ein leeres Element sein.

Datei **checklist.dtd**:

Aufgabe 5: (9)

Betrachten Sie dieses XSLT-Stylesheet **query.xsl**:

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

    <!-- Platzhalter -->

</xsl:stylesheet>
```

Im folgenden finden Sie sechs Möglichkeiten, wie der Kommentar `<!-- Platzhalter -->` ersetzt werden kann. Sie können davon ausgehen, dass das XSLT-Stylesheet syntaktisch korrekt bleibt, d.h. konzentrieren Sie sich auf die Funktionalität.

Kreuzen Sie an, für welche der Ersetzungen des Kommentars `<!-- Platzhalter -->` die Ausgabe des Stylesheets, angewandt auf **checklist.xml**, folgenden Text **enthält**:

```
Arbeitszeit
```

Beispielsweise wäre für folgende Ausgabe die Antwort "ja":

```
<li>Arbeitszeit</li>
<li>Abgabemodalitaeten</li>
```

und für folgende Ausgabe die Antwort "nein":

```
<list id="list2"/>
```

Die exakte Behandlung von Whitespaces etc. ist für diese Beispiel nicht relevant.

1.
```
<xsl:template match="list[@id='list3']">
    <xsl:copy/>
</xsl:template>
```
ja ◯ nein ◯

2.
```
<xsl:template match="list[@id='list3']">
    <xsl:copy-of select="."/>
</xsl:template>
```
ja ◯ nein ◯

3.
```
<xsl:template match="/">
    <xsl:value-of select="text()"/>
</xsl:template>
```
ja ◯ nein ◯

4.
```
<xsl:template match="text()"/>
```
ja ◯ nein ◯

5.
```
<xsl:template match="checklist">
    <xsl:copy-of select="list//li"/>
</xsl:template>
```
ja ◯ nein ◯

6.
```
<xsl:template match="li">
    <xsl:if test="position() = 1">
        <xsl:value-of select="."/>
    </xsl:if>
    <xsl:apply-templates select="*"/>
</xsl:template>
```
ja ◯ nein ◯

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrekter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Aufgabe 6: (10)

Betrachten Sie die folgenden XPath-Abfragen angewandt auf das Dokument **checklist.xml** (siehe Anhang).

- Verwenden sie als Kontextknoten das `list` Element mit dem `id` Attributwert `list3`.
- Geben Sie als Ausgabe die Werte der `id` Attribute an.
- Falls der angegebene XPath Ausdruck keine Knoten selektiert, notieren Sie im entsprechenden Feld "leere Ausgabe".

Betrachten Sie dazu folgendes Beispiel:

`descendant-or-self::list`

`list3 list4`

Geben Sie nun die `id` Attributwerte der von den folgenden XPath-Abfragen ausgewählten `list` Elemente an.

`ancestor::list`

`.//list`

`li/..`

`following-sibling::list`

`following::list`

Aufgabe 7: (8)

Erstellen Sie eine XQuery-Anfrage **query.xq**, die angewandt auf Dokumente der Gestalt **checklist.xml** folgende Ausgabe liefert:

- Für jedes Element `list` soll die Anzahl der `li` Kindelemente berechnet werden.
- Diese Anzahl soll in der folgenden Form ausgegeben werden:

  <listen-id>anzahl</listen-id>

  d.h. der Elementname entspricht dem Wert des `id` Attribut des jeweiligen `list` Elements.
- Es sollen nur jene `list` Elemente ausgegeben werden, die mehr als ein `li` Kindelement enthalten.
- Sortieren Sie die Ausgabe entsprechend dem Wert des `id` Attributs in absteigender Reihenfolge.

Beispielsweise ist die gewünschte Ausgabe, angewandt auf **checklist.xml**:

```
<list5>3</list5>
<list4>3</list4>
<list2>2</list2>
<list1>3</list1>
```

Geben Sie hier die XQuery-Anfrage **xquery.xq** an:

Datei **xquery.xq**:

Aufgabe 8: (9)

Erstellen Sie einen SAX `ContentHandler`, der die (Verschachtelungs-)Tiefe eines XML Dokuments berechnet.

Beispielsweise hat folgendes Dokument die Tiefe `1`:

```
<a/>
```

und folgendes Dokument hat die Tiefe `3`:

```
<a>
    <b>
        <c/>
    </b>
    <d/>
</a>
```

D.h. die Tiefe eines XML Dokuments entspricht dem Maximum der Anzahl von Ahnenelementen, die ein Element des Dokuments besitzt plus 1.

Vervollständigen Sie nun den folgenden Java Code um die lokalen Variablen und Eventhandler-Methoden, die sie benötigen (um Imports, Fehlerbehandlung, etc. brauchen Sie sich nicht kümmern):

```
public class DepthHandler extends DefaultHandler {
    int docDepth;

    public void endDocument() throws SAXException {
        System.out.println("Document Depth: " + docDepth);
    }
}
```

Sie können diese Seite abtrennen!

Datei **checklist.xml**:

```
<checklist>
   <list titel="CHECKLIST SSD-Pruefung" id="list1">
      <li>
         <i>Vor</i> der Pruefung sind folgende Sachen zu tun:
         <list id="list2">
            <li>Hoersaal reservieren</li>
            <li>Anmeldung freischalten</li>
         </list>
      </li>
      <hline/>
      <li>Merklist zur Pruefung selbst:
         <list id="list3">
            <li>
               Spielregeln vorlesen
               <list id="list4">
                  <li><b>Studierendenausweis bereithalten</b></li>
                  <li>Arbeitszeit</li>
                  <li>Abgabemodalitaeten</li>
               </list> und Angaben austeilen
            </li>
         </list>
      </li>
      <hline/>
      <li>Nach der Pruefung:
         <list titel="CHECKLIST Korrektur" id="list5">
            <li>korrigieren</li>
            <li>ins TISS spielen</li>
            <li><i>Einsichtnahmetermin</i> fixieren und
                Notenvorschlag ausschicken!</li>
         </list>
      </li>
   </list>
</checklist>
```

Gesamtpunkte: 75