Gruppe A	PRÜFUNG AUS "SEMISTRUKTURIERTE DATEN" **184.705**		26. 06. 2013
Kennnr.	Matrikelnr.	Familienname	Vorname

Arbeitszeit: 100 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: (12)

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

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

    <xsd:element name="a" type="atype"/>

    <xsd:complexType name="atype">
        <xsd:sequence>
            <xsd:element name="b" type="btype" minOccurs="0" maxOccurs="2"/>
            <xsd:element name="a" type="atype" minOccurs="0" maxOccurs="2"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="btype">
        <xsd:choice minOccurs="0" maxOccurs="1">
            <xsd:element name="c" type="xsd:int"/>
            <xsd:element name="a" type="atype"/>
        </xsd:choice>
    </xsd:complexType>
</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/>` gültig ⊗ ungültig ◯
2. `<b/>` gültig ◯ ungültig ⊗
3. `<a><a/><a/></a>` gültig ⊗ ungültig ◯
4. `<a><a/><a/><a/><a/></a>` gültig ◯ ungültig ⊗
5. `<a><b><c>1</c></b></a>` gültig ⊗ ungültig ◯
6. `<a><b><a/></b></a>` gültig ⊗ ungültig ◯
7. `<a><b><a><b/></a></b></a>` gültig ⊗ ungültig ◯
8. `<a><b><a><c>1</c></a></b></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: (15)

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

1.	XML gibt es schon wesentlich länger als HTML.	wahr ◯ falsch ⊗
2.	Um die Wohlgeformtheit eines XML-Dokuments zu überprüfen wird eine DTD oder ein XML Schema benötigt.	wahr ◯ falsch ⊗
3.	Eine XML Schema Definition muss ein wohlgeformtes XML-Dokument sein.	wahr ⊗ falsch ◯
4.	Pro XML-Schema Datei ist maximal ein Target-Namespace erlaubt.	wahr ⊗ falsch ◯
5.	SAX ist keine W3C-Recommendation, aber ein de-facto Standard.	wahr ⊗ falsch ◯
6.	SAX Filter dürfen nicht verschachtelt verwendet werden.	wahr ◯ falsch ⊗
7.	Der Speicherbedarf eines DOM-Parsers ist unabhängig von der Größe des geparsten XML-Files.	wahr ◯ falsch ⊗
8.	In XPath gilt `(1,2) = (2,3)`	wahr ⊗ falsch ◯
9.	In XPath gilt `(1,2) != (2,3)`	wahr ⊗ falsch ◯
10.	XSLT Stylesheets sind wohlgeformte XML Dokumente.	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 3 – 7 beziehen sich auf das XML-Dokument struktur.xml, das Sie auf der letzten Seite dieser Prüfungsangabe finden.

Aufgabe 3: (12)

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

- Das Element `struktur` ist das Wurzelelement und besteht aus genau einem `knoten`-Element gefolgt von zumindest einem `typ`-Element.
- Das Element `knoten` besteht aus beliebig vielen `inhalt` Elementen gefolgt von beliebig vielen `knoten` Elementen. Der Wert des Attributs `id` soll global eindeutig sein.
- Das Element `inhalt` besitzt Textinhalt. Das Attribut `typ` der `inhalt`-Elemente verweist auf das Primärschlüssel-Attribut `name` des `typ`-Elements.
- Das Element `typ` besitzt leeren Inhalt und drei Attribute.
- Spezifizieren Sie die Attribute (wählen Sie sinnvolle Typen aus!) sowie die nicht näher erläuterten Häufigkeiten entsprechend dem **struktur.xml** Dokument im Anhang. Beachten Sie auch die Rekursion der `knoten`-Elemente. Alle Attribute sollen verpflichtend sein. Versuchen Sie weiters entsprechende Fremdschlüssel zu finden und in der DTD abzubilden.

Datei **struktur.dtd**:

```
<!ELEMENT struktur (knoten, typ+)>

<!ELEMENT knoten (inhalt*, knoten*)>
<!ATTLIST knoten
   id ID #REQUIRED
>

<!ELEMENT inhalt (#PCDATA)>
<!ATTLIST inhalt
   typ IDREF #REQUIRED
>

<!ELEMENT typ EMPTY>
<!ATTLIST typ
   name ID #REQUIRED
   min CDATA #REQUIRED
   max CDATA #REQUIRED
>
```

Aufgabe 4: (10)

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

- Falls als Ergebnis `knoten` Elemente selektiert werden, geben Sie jeweils das Attribut `id` an.
- Falls als Ergebnis mehrere Elemente selektiert werden, trennen Sie die jeweiligen Ausgaben durch Leerzeichen.
- Falls der angegebene XPath Ausdruck keine Knoten selektiert, notieren Sie im entsprechenden Feld "leere Ausgabe".

Betrachten Sie dazu folgendes Beispiel:

```
//knoten
```

a b c d e f g

Geben Sie nun die entsprechende Ausgaben der folgenden XPath-Abfragen an.

```
//knoten[not(*)]
```

g

```
//knoten[ancestor::knoten]
```

b c d e f g

```
//knoten[2]
```

c

```
(//knoten)[2]
```

b

```
//knoten[knoten[.//@typ='int']]
```

a c

Aufgabe 5: (8)

Betrachten Sie folgende-XQuery Abfrage **struktur.xq** angewandt auf **struktur.xml**:

```
let $x := //knoten[sum(inhalt) > 0]
for $y in //knoten[inhalt/@typ='int']
return
   <k id="{$y/@id}">{
      if ($y = $x)
      then attribute b {"true"}
      else <pm/>,
      sum($y/inhalt)
   }</k>
```

Geben Sie zuerst die `id` Attribute aller Knoten an, die in `$x` gespeichert sind:

```
d e
```

Geben Sie nun die `id` Attribute aller Knoten an, die `$y` durchläuft:

```
a b d
```

Geben Sie nun die Ausgabe von **struktur.xq** angewandt auf **struktur.xml** an.
Die exakte Behandlung von Whitespaces ist dabei nicht relevant.

```
<k id="a"><pm/>-1</k>
<k id="b"><pm/>0</k>
<k id="d" b="true">6</k>
```

Aufgabe 6: (9)

Erstellen Sie ein XSLT-Stylesheet **struktur.xsl**, das angewandt auf Dokumente der Gestalt **struktur.xml** folgende Ausgabe erzeugt:

- Als Dokumentelement soll das Element `out` erzeugt werden.
- Für jedes `knoten` Element, das weitere `knoten` Elemente enthält, wird ein Element `inner` erzeugt.
- Für jedes Kindelement `inhalt` des Typs `pos` oder `int`, wird ein `val` Element erzeugt, das dessen Textinhalt enthält.

Für das Dokument **struktur.xml** soll beispielsweise folgende Ausgabe erzeugt werden:

```
<out>
  <inner>
    <val>2</val>
    <inner/>
    <inner>
       <val>5</val>
    </inner>
  </inner>
</out>
```

Vervollständigen Sie hier das XSLT-Stylesheet **struktur.xsl**. Die Verwendung von Kontrollstrukturen wie `xsl:for-each`, `xsl:if`, etc. ist für die Lösung *nicht* erlaubt (und auch nicht sinnvoll)! Sie brauchen sich nicht um Whitespaces etc. zu kümmern.

Datei **struktur.xsl**:

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

   <xsl:template match="/">
      <out>
         <xsl:apply-templates/>
      </out>
   </xsl:template>

   <xsl:template match="knoten[knoten]">
      <inner>
         <xsl:apply-templates/>
      </inner>
   </xsl:template>

   <xsl:template match="inhalt[@typ='pos' or @typ='int']">
      <xsl:value-of select="."/>
   </xsl:template>

  <xsl:template match="text()|knoten"/>

</xsl:stylesheet>
```

Aufgabe 7: (9)

Vervollständigen Sie den folgenden SAX Handler, der angewandt auf Dokumente der Gestalt **struktur.xml** die Summe aller `inhalt` Elemente des Typs `pos` berechnet. Die Summe soll in die Variable `sum` geschrieben werden.

Für das Dokument **struktur.xml** wird beispielsweise folgende Summe erreicht: `9` (= `4` + `5`)

Denken Sie daran, dass `character` Events nicht immer den gesamten Textinhalt auf einmal zurückliefern!

```
public class PosSum extends DefaultHandler {

  int sum = 0;
  boolean inPos = false;
  StringBuffer sb = new StringBuffer();

  public void startElement(String uri, String localName, String qName, Attributes atts)
       throws SAXException {
    if ("pos".equals(atts.getValue("typ")) {
     inPos = true;
    }
  }

  public void endElement(String uri, String localName, String qName) throws SAXException {
    inPos = false;
    sum += Integer.parseInt(sb.toString());
    sb.delete(0, sb.length());
  }

  public void characters(char[] ch, int start, int length) throws SAXException {
     if (inPos) {
      sb.append(ch, start, length);
     }
  }
}
```

Sie können diese Seite abtrennen!

Datei **struktur.xml**:

```
<struktur>
   <knoten id="a">
      <inhalt typ="int">2</inhalt>
      <inhalt typ="neg">-3</inhalt>

      <knoten id="b">
         <inhalt typ="pos">4</inhalt>
         <inhalt typ="int">-4</inhalt>
      </knoten>

      <knoten id="c">
         <knoten id="d">
            <inhalt typ="int">6</inhalt>
         </knoten>
      </knoten>

      <knoten id="e">
         <inhalt typ="pos">5</inhalt>
         <knoten id="f">
            <inhalt typ="neg">-9</inhalt>
         </knoten>
      </knoten>

      <knoten id="g"/>
   </knoten>

   <typ name="int" min="-15" max="16"/>
   <typ name="neg" min="-32" max="-1"/>
   <typ name="pos" min="1" max="32"/>
</struktur>
```

Gesamtpunkte: 75