Gruppe A	PRÜFUNG AUS "SEMISTRUKTURIERTE DATEN" **184.705**		24. 10. 2014
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 DTD Datei **test.dtd**:

```
<!ELEMENT X ((X|Y),(Y|Z))>
<!ELEMENT Y (#PCDATA | Z)*>
<!ELEMENT Z EMPTY>
<!ATTLIST X ref IDREF #IMPLIED>
<!ATTLIST Z key ID #REQUIRED>
```

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.dtd** zu entscheiden.

Kreuzen Sie an, welche der folgenden XML-Dateien gültig bezüglich **test.dtd** sind (nehmen Sie an, dass **X** als Wurzelelement spezifiziert ist):

1. `<X/>` gültig ◯ ungültig ⊗
2. `<X><Y/><Y/></X>` gültig ⊗ ungültig ◯
3. `<X><Y/><Y><Z key="a"/></Y></X>` gültig ⊗ ungültig ◯
4. `<X><Y/><Y><Z key="a"/><Z key="a"/></Y></X>` gültig ◯ ungültig ⊗
5. `<X><Y><Z key="a"/></Y><Y><Z key="a"/></Y></X>` gültig ◯ ungültig ⊗
6. `<X ref="a"><Y>abc</Y><Z key="a"/></X>` gültig ⊗ ungültig ◯
7. `<X ref="b"><Y>abc</Y><Z key="a"/></X>` gültig ◯ ungültig ⊗
8. `<X><X><Y/><Y/></X><Y/></X>` 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. HTML ist eine Weiterentwiclung von XML. wahr ◯ falsch ⊗
2. Ein (bezüglich eines Schemas) gültiges XML-Dokument muss auch wohlgeformt sein. wahr ⊗ falsch ◯
3. DTDs erlauben typischerweise eine exaktere Spezifikation als XML Schema. wahr ◯ falsch ⊗
4. Pro XML-Schema Datei ist maximal ein Target-Namespace erlaubt. wahr ⊗ falsch ◯
5. In XPath gilt `(a,b) != (b,c)`. wahr ⊗ falsch ◯
6. SAX Filter dürfen verschachtelt verwendet werden. wahr ⊗ falsch ◯
7. SAX-Parser benutzen intern einen DOM-Parser. wahr ◯ falsch ⊗
8. Bei XSLT wird pro Knoten maximal ein Template angewendet. wahr ⊗ falsch ◯
9. XQUERY ist eine W3C Recommendation. wahr ⊗ falsch ◯
10. JSON ist eine XML-Applikation. 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 kontrollgraph.xml, das Sie auf der letzten Seite dieser Prüfungsangabe finden.

Aufgabe 3: (12)

Vervollständigen Sie die XML Schema Datei **kontrollgraph.xsd**, sodass XML-Dokumente in der Gestalt von **kontrollgraph.xml** (siehe Anhang) bezüglich dieses Schemas gültig sind. Berücksichtigen Sie beim Erstellen des Schemas folgende Punkte:

- Das Element `code` beinhaltet beliebig viele `if` Elemente sowie beliebigen Text, und kann mit einen Attribut `id` versehen sein. Andererseits kann es als leeres Element mit einem Attribut `ref` verwendet werden (letzteres soll GOTO-Anweisungen im Code abbilden).

  Hinweis: Versuchen Sie die Struktur des `code`-Elements, so gut wie möglich abzubilden. Eine exakte Spezifikation (falls das Attribut `ref` auftritt soll das Element leer sein) ist nur sehr umständlich zu realisieren und daher nicht gefordert.

- Elemente `if` haben ein verpflichtendes Attribut `cond` und 1 oder 2 Kindelemente `code`.

- Definieren Sie die Abhängigkeiten der Attributwerte `ref` und `id` als Schlüsselbeziehung.

  Hinweis: Da Attribut `id` nicht verpflichtend ist, kann das `xsd:key` Element hier nicht angewendet werden!

- Sollten bei bestimmten Elementen oder Attributen keine näheren Angaben bezüglich des genauen Typs vorgegeben sein, wählen Sie selbst einen sinnvollen Typ aus.

Datei **kontrollgraph.xsd**:

```
<!-- Sie haben auch noch auf der folgenden Seite Platz! -->

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

   <xsd:element name="code" type="codeType">
      <xsd:unique name="codeKey">
         <xsd:selector xpath=".//code"/>
         <xsd:field xpath="@id"/>
      </xsd:unique>

      <xsd:keyref name="codeKeyRef" refer="codeKey">
         <xsd:selector xpath=".//code"/>
         <xsd:field xpath="@ref"/>
      </xsd:keyref>
   </xsd:element>

   <xsd:complexType name="codeType" mixed="true">
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
         <xsd:element name="if" type="ifType"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:string" use="optional"/>
      <xsd:attribute name="ref" type="xsd:string" use="optional"/>
   </xsd:complexType>

   <xsd:complexType name="ifType">
      <xsd:sequence minOccurs="1" maxOccurs="2">
         <xsd:element name="code" type="codeType"/>
      </xsd:sequence>
      <xsd:attribute name="cond" type="xsd:string" use="required"/>
   </xsd:complexType>

</xsd:schema>
```

Datei **kontrollgraph.xsd** (Fortsetzung):

Aufgabe 4: (10)

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

- Falls als Ergebnis mehrere Knoten selektiert werden, trennen Sie die jeweiligen Ausgaben durch Leerzeichen.
- Falls der angegebene XPath Ausdruck keine Knoten selektiert, notieren Sie im entsprechenden Feld "leere Ausgabe".
- Falls als Ergebnis `code` Knoten selektiert werden, geben Sie deren `id` Attribute an.
- Falls als Ergebnis eine Zahl ausgegeben wird, geben Sie diese an.

Betrachten Sie dazu folgendes Beispiel:

```
//code
```

```
fehler main even odd
```

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

```
count(//code)
```

```
9
```

```
//code[2]
```

```
main
```

```
count(//code/@ref)
```

```
2
```

```
//code[if]
```

```
main
```

```
count(//code[@id='main']/*)
```

```
4
```

Aufgabe 5: (8)

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

```
for $id in //code/@id
let $count := count(//code[@ref=$id])
where $count < 10
return element {$id} {$count}
```

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

```
<main>1</main>
<fehler>0</fehler>
<even>0</even>
<odd>1</odd>
```

Aufgabe 6: (10)

Erstellen Sie ein XSLT-Stylesheet **kontrollgraph.xsl**, das angewandt auf Dokumente der Gestalt **kontrollgraph.xml** Code im C/Java-Stil erzeugt. Für das Dokument **kontrollgraph.xml** soll beispielsweise folgende Ausgabe erzeugt werden:

```
if (x=0) {
    ausgabe: fehler
} else {
    ausgabe: ok
    if (x=1) {
       ausgabe: odd number
    }
    if (x=2) {
        ausgabe: even number
    }
    if (x=3) {
        ausgabe: odd number
    } else {
        exit
    }
    ...
}
```

Das bedeutet also:

- `if` Elemente sollen durch folgenden Text ersetzt werden:

  `if (`*condition*`) {` *erster Zweig* `} else {` *zweiter Zweig* `}`

  wobei der Text `else {`*zweiter Zweig* `}` nur ausgegeben werden soll, wenn auch wirklich ein zweiter Zweig vorhanden ist (d.h., ein zweites `code` Kindelement).
- Bei `code` Elementen mit `ref` Attribut soll die Verarbeitung beim `code` Element mit dem entsprechenden `id` Attribut fortgesetzt werden.

Vervollständigen Sie hier das XSLT-Stylesheet **kontrollgraph.xsl**. Sie brauchen sich nicht um Whitespaces etc. zu kümmern.

Datei **kontrollgraph.xsl**:

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text" version="1.0" encoding="UTF-8"/>

    <xsl:template match="code[not(@ref)]">
        <xsl:apply-templates/>
    </xsl:template>

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

    <xsl:template match="if">
        if ( <xsl:value-of select="@cond"/> ) {
            <xsl:apply-templates select="code[1]"/>
        }

        <xsl:if test="code[2]">
           else { <xsl:apply-templates select="code[2]"/> }
        </xsl:if>
    </xsl:template>
</xsl:stylesheet>
```

Aufgabe 7: (8)

Vervollständigen Sie den folgenden SAX Handler, der angewandt auf Dokumente der Gestalt **kontrollgraph.xml** prüfen soll, ob es im Dokument Vorwärtsreferenzen gibt.

- Eine Vorwärtsreferenz ist ein `code` Element mit `ref` Attribut, das im Dokument früher (in Document Order) vorkommt als das `code` Element mit dem entsprechenden `id` Attribut.

  Beispielsweise kommt im Dokument **kontrollgraph.xml** das Element `<code ref="odd">...` früher vor als das Element `<code id="odd">...` und ist daher eine Vorwärtsreferenz.
- Für jede gefunden Vorwärtsreferenz soll der Wert des `ref` Attributs ausgegeben werden.

Für das Dokument **kontrollgraph.xml** wird beispielsweise folgende Meldung ausgegeben:

```
Vorwärtsreferenz: odd
```

Um die genaue Formatierung der Ausgabe brauchen Sie sich nicht zu kümmern

```
public class CheckForwardReferences extends DefaultHandler {

  Set ids = new Set<String>();

  public void startElement(String uri, String localName, String qName, Attributes atts)
      throws SAXException {
    if ("code".equals(localName)) {
        if (atts.getIndex("id") != -1) {
          ids.put(atts.getValue("id"));
        }
        if (atts.getIndex("ref") != -1 && !ids.contains(atts.getValue("ref"))) {
          System.out.println("Vorwärtsreferenz: " + atts.getValue("ref"));
        }
    }
  }
}
```

Sie können diese Seite abtrennen!

Datei **kontrollgraph.xml**:

```
<code>
   <if cond="x=0">
      <code id="fehler">ausgabe: fehler</code>
      <code id="main">
         ausgabe: ok
         <if cond="x=1">
            <code ref="odd"/>
         </if>
         <if cond="x=2">
            <code id="even">ausgabe: even number</code>
         </if>
         <if cond="x=3">
            <code id="odd">ausgabe: odd number</code>
            <code>exit</code>
         </if>
         <if cond="true">
            <code>exit</code>
            <!-- (never) do the loop -->
            <code ref="main"/>
         </if>
      </code>
   </if>
</code>
```

Gesamtpunkte: 75