PRÜFUNG IN "SEMI-STRUKTURIERTE DATEN" **184.705**			09. 01. 2018
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 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 maxOccurs="2">
                <xsd:element name="B" minOccurs="0" type="xsd:integer"/>
                <xsd:element name="C" type="typeC"/>
            </xsd:choice>
        </xsd:complexType>
        <xsd:key name="key1">
            <xsd:selector xpath=".//C"/>
            <xsd:field xpath="@id"/>
         </xsd:key>
    </xsd:element>

    <xsd:complexType name="typeC" mixed="false">
         <xsd:sequence>
                <xsd:element name="C" type="typeC" minOccurs="0" maxOccurs="2"/>
          </xsd:sequence>
          <xsd:attribute name="id" type="xsd:integer"/>
    </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>SSD</A>` valid ⊗ invalid ◯
2. `<A>DBAI<B>1842</B></A>` valid ⊗ invalid ◯
3. `<A><C/></A>` valid ◯ invalid ⊗
4. `<A><C id="3"><C id="1"><C id="4"></C></C></C></A>` valid ⊗ invalid ◯
5. `<A><C id="1"><C id="4"><C id="1"></C></C></C></A>` valid ◯ invalid ⊗
6. `<A><B>42</B><C id="1"/></A>` valid ⊗ invalid ◯
7. `<A><B>42</B><C id="1"/><B>23</B></A>` valid ◯ invalid ⊗
8. `<C></C>` valid ◯ invalid ⊗

(Für jede korrekte Antwort 1.5 Punkte, **für jede falsche Antwort -1.5 Punkte**, unbeanwortete Fragen 0 Punkt, Insgesamt nicht weniger als 0 Punkte)

Aufgabe 2: (15)

Entscheiden Sie, ob die folgenden Aussagen wahr oder falsch sind.

1. In einem wohlgeformten XML-Dokument dürfen zwei Attribute mit demselben qualifizierten Namen in einem Elementstarttag auftreten — wahr ◯ falsch ⊗
2. Genauso wie bei Elementen ist die Reihenfolge von Attributen innerhalb eines Elements in XML-Dokumenten signifikant. — wahr ◯ falsch ⊗
3. DOM erlaubt wahlfreien Zugriff auf das gesamte XML Dokument. — wahr ⊗ falsch ◯
4. Frage entfernt (da nicht eindeutig)
5. In XPath beginnt ein relativer Pfad immer vom aktuellen Context Node aus. — wahr ⊗ falsch ◯
6. Wenn in XSLT für einen Knoten kein Template angegeben wird, dann wird die Verarbeitung abgebrochen, falls dieser in der Eingabe auftritt. — wahr ◯ falsch ⊗
7. Das Ergebnis eines XSLT-Stylesheets ist immer ein wohlgeformtes XML Dokument. — wahr ◯ falsch ⊗
8. DTD steht für Document Type Definition. — wahr ⊗ falsch ◯
9. Um die Wohlgeformtheit eines XML-Dokuments zu überprüfen wird eine DTD oder ein XML-Schema benötigt. — wahr ◯ falsch ⊗
10. Attribute ohne Prefix sind im Default Namespace. — wahr ◯ falsch ⊗

(Für jede korrekte Antwort 1.5 Punkte, **für jede falsche Antwort -1.5 Punkte**, unbeanwortete Fragen 0 Punkt, Insgesamt nicht weniger als 0 Punkte)

Die folgenden Aufgaben 3 – 7 beziehen sich auf das XML-Dokument christmas.xml, das Sie auf der letzten Seite dieser Prüfungsangabe finden.

Aufgabe 3: (12)

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

- Das `gifts` Element enthält beliebig viele `gift` Elemente. Das `persons` Element enthält beliebig viele `person` Elemente.
- `gift` Elemente enthalten ein `name` Element, möglicherweise ein `description` Element, beliebig viele `from` Elemente, mindestens ein `to` Element und ein `price` Element (in dieser Reihenfolge).
- `person` Elemente enthalten ein `name` Element und möglicherweise ein `budget` Element.
- `gift` und `person` Elemente haben als Attribut eine eindeutige ID.
- Das Attribut `pid` in den `from` und `to` Elementen verweist immer auf die ID eines `person` Elements.
- Wenn nicht angegeben treffen Sie plausible Annahmen über Typen von Attributen und Elementen.

File **christmas.dtd**:

```
<!ELEMENT christmas (gifts,persons)>

<!ELEMENT gifts (gift*)>
<!ELEMENT persons (person*)>

<!ELEMENT gift (name,description?,from*,to+,price)>
<!ATTLIST gift nr ID #REQUIRED>

<!ELEMENT person (name,budget?)>
<!ATTLIST person pid ID #REQUIRED>

<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>

<!ELEMENT from EMPTY>
<!ATTLIST from pid IDREF #REQUIRED>
<!ELEMENT to EMPTY>
<!ATTLIST to pid IDREF #REQUIRED>

<!ELEMENT price (#PCDATA)>
<!ELEMENT budget (#PCDATA)>
```

Aufgabe 4: (10)

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

- Falls der angegebene XPath Ausdruck keine Knoten selektiert, notieren Sie im entsprechenden Feld "leere Ausgabe".
- Falls als Ergebnis eine Zahl selektiert wird (count, sum, ...), geben Sie diese Zahl an.

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

```
min(//gift/price)
```

```
40
```

```
//person[@pid=//gift[count(from)>1]/to/@pid]/name
```

```
<name>Alice</name>
```

```
//gift[description][last()]/name/text()
```

```
Swiss Army Knife
```

```
//gift[last()][description]/name/text()
```

```
leere Ausgabe
```

```
//person[not(@pid=//from/@pid)]/name/text()
```

```
Findus
Alice
Bob
```

Aufgabe 5: (10)

Schreiben Sie eine XQuery Abfrage um folgende Aufgabenstellung zu lösen:

Es soll ein XML Dokument mit der Wurzel `persons` erzeugt werden.

In diesem Dokument sollen alle Personen ausgegeben werden, die zumindest ein Geschenk verschenkt haben (es gibt zumindest ein `gift` Element dessen `from` Kind auf die Person verweist). Ordnen Sie die Personen nach deren Namen absteigend!

Für jede Person werden die Namen aller erhaltenen Geschenke in `gift` Elementen ausgegeben. Ordnen Sie die erhaltenen Geschenke nach deren Namen absteigend!

Zusätzlich soll für jede Person die Summe der Preise der erhaltenen Geschenke minus die Summe der Preise der verschenkten Geschenke in einem `summary` Element ausgegeben werden.

Ihre Abfrage soll, angewandt auf **christmas.xml**, folgende Ausgabe liefern:

```
<persons>
    <person name="Father">
        <gift>Skiing day with the family</gift>
        <summary>190</summary>
    </person>
    <person name="Mum">
        <gift>Skiing day with the family</gift>
        <summary>-180</summary>
    </person>
</persons>
```

Geben Sie nun die XQuery Abfrage an:

```
<persons>
{
   for $p in //person
   where count(//gift[from/@pid = $p/@pid]) >= 1
   order by $p/name ascending
   return
      <person name="{$p/name}">
      {
         for $g in //gift[to/@pid = $p/@pid]
         order by $g/name
         return
            <gift>{$g/name/text()}</gift>
      }
        <summary>
          {sum(//gift[to/@pid = $p/@pid]/price)-sum(//gift[from/@pid = $p/@pid]/price)}
        </summary>
      </person>
}
</persons>
```

Aufgabe 6: (8)

Betrachten Sie folgendes XSLT Dokument **christmas.xsl**:

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

    <xsl:template match="christmas">
        <xsl:apply-templates select="//gift" />
    </xsl:template>

    <xsl:template match="gift">
        <xsl:apply-templates select="from" />
    </xsl:template>

    <xsl:template match="from">
        <xsl:variable name="fpid" select="@pid" />
        <xsl:variable name="gname" select="../name" />
        <xsl:for-each select="../to">
            <xsl:apply-templates select="//person[@pid=$fpid]" />
            --(<xsl:value-of select="$gname" />)-->
            <xsl:apply-templates select="//person[@pid=current()/@pid]" />
        </xsl:for-each>
    </xsl:template>

    <xsl:template match="person">
        <xsl:value-of select="name" />
    </xsl:template>

</xsl:stylesheet>
```

Geben Sie nun die Ausgabe von **christmas.xsl** angewandt auf **christmas.xml** an.

Sie müssen sich nicht um Whitespaces kümmern.

```
Mum --(cat tree)--> Findus
Mum --(NES)--> Alice
Father --(NES)--> Alice
Father --(MacGyver: The Complete Collection)--> Bob
Mum --(Skiing day with the family)--> Father
Mum --(Skiing day with the family)--> Mum
Mum --(Skiing day with the family)--> Alice
Mum --(Skiing day with the family)--> Bob
```

Aufgabe 7: (8)

Vervollständigen Sie die Methode `insPresent`, die drei Parameter `gnr`, `fpid` und `tpid` besitzt und mittels **DOM** folgende Änderungen am, in der Variable `doc` gespeicherten, Dokument durchführt:

> Im `gift` mit der Nr `gnr` wird an korrekter Stelle ein `from` Element mit der `pid` `fpid` und ein `to` Element mit der `pid` `tpid` eingefügt.

Dazu gehen Sie wie folgt vor:

- Sie suchen mit Hilfe von `xPath` das `gift` Element mit der Nr `gnr`. Falls ein solches nicht vorhanden ist, werfen Sie mittels `throw new Exception();` eine Exception.
- Sie finden das erste `to` Kindelement des `gift` Elements.
- Sie erstellen ein neues `from` und ein neues `to` Element mit dem Attribut `pid` gesetzt auf `fpid` bzw. `tpid`.
- Sie fügen die neuen Elemente vor dem gefunden `to` Kindelement ein.

```
private static XPath xPath = XPathFactory.newInstance().newXPath();
Document doc;

public void insPresent (String gnr, String fpid, String tpid) throws Exception {
    XPathExpression xpe = xPath.compile("//gift[@nr = '" + gnr + "']");

    NodeList list = (NodeList) xpe.evaluate(doc, XPathConstants.NODESET);

    if (list.getLength() == 1) {
        //find first to element
        Node gift = list.item(0);
        Node refTo = gift.getFirstChild();
        while (!refTo.getNodeName().equals("to"))
            refTo = refTo.getNextSibling();

        //create new elements
        Element newFrom = doc.createElement("from");
        newFrom.setAttribute("pid", fpid);
        Element newTo = doc.createElement("to");
        newTo.setAttribute("pid", tpid);

        //insert from and to before first to element
        gift.insertBefore(newFrom, refTo);
        gift.insertBefore(newTo, refTo);
    } else
        throw new Exception("Keine gültige Gift ID!");

}
```

Total points: 75

File **christmas.xml**: **You may separate this page!**

```
<christmas>
  <gifts>
      <gift nr="g1">
          <name>cat tree</name>
          <description>Throne for the boss</description>
          <from pid="p3"/>
          <to pid="p1"/>
          <price>110</price>
      </gift>
      <gift nr="g2">
          <name>NES</name>
          <description>8-bit home video game console</description>
          <from pid="p3"/>
          <from pid="p2"/>
          <to pid="p4"/>
          <price>70</price>
      </gift>
      <gift nr="g3">
          <name>MacGyver: The Complete Collection</name>
          <description>The original Series</description>
          <from pid="p2"/>
          <to pid="p5"/>
          <price>40</price>
      </gift>
      <gift nr="g4">
          <name>Swiss Army Knife</name>
          <description>The Right Tool for the Task</description>
          <to pid="p5"/>
          <price>53</price>
      </gift>
      <gift nr="g5">
          <name>Skiing day with the family</name>
          <from pid="p3"/>
          <to pid="p2"/>
          <to pid="p3"/>
          <to pid="p4"/>
          <to pid="p5"/>
          <price>300</price>
      </gift>
  </gifts>
  <persons>
    <person pid="p1">
        <name>Findus</name>
    </person>
    <person pid="p2">
        <name>Father</name>
        <budget>450</budget>
    </person>
    <person pid="p3">
        <name>Mum</name>
        <budget>550</budget>
    </person>
    <person pid="p4">
        <name>Alice</name>
    </person>
    <person pid="p5">
        <name>Bob</name>
    </person>
  </persons>
</christmas>
```