| Gruppe A | PRÜFUNG AUS "SEMISTRUKTURIERTE DATEN" **181.135** | | 05. 12. 2011 |
|---|---|---|---|
| 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 DTD Datei **test.dtd**:

```
<!ELEMENT A ((B, C, D) | (A)*)>
<!ELEMENT B (#PCDATA | C)*>
<!ELEMENT C ANY>
<!ELEMENT D EMPTY>
<!ATTLIST A key ID #REQUIRED>
<!ATTLIST D ref IDREF #IMPLIED>
```

Betrachten Sie weiters die sechs 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.

1. `<A key="a1"><B>random</B><C/><D ref="a1"/></A>` gültig ⊗ ungültig ◯
2. `<A key="a1"><B/><C><A key="a2"><B/><C/><D ref="a1"/></A>text</C><D/></A>` gültig ⊗ ungültig ◯
3. `<A><B/><C/><D/></A>` gültig ◯ ungültig ⊗
4. `<A key="a1"><B><C><C/><B/></C></B><C/><D/></A>` gültig ⊗ ungültig ◯
5. `<A key="a1"><B>text</B><C><A key="a2"><B/><D ref="a1"/></A></C><D/></A>` gültig ◯ ungültig ⊗
6. `<A key="a1"><B/><C><D ref="a1"/></C><D ref="a2"/></A>` gültig ◯ ungültig ⊗

(Regeln für Beispiele 1–3: 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:** (9)

Betrachten Sie die folgende Schema-Datei **ns.xsd**:

```
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.dbai.tuwien.ac.at/education/ssd/SS11/pruefung/NS"
    xmlns:t="http://www.dbai.tuwien.ac.at/education/ssd/SS11/pruefung/NS" elementFormDefault="qualified">

    <xs:element name="root-node" type="t:nodeType">
        <xs:key name="key">
            <xs:selector xpath=".//*"/>
            <xs:field xpath="@t:id"/>
        </xs:key>
        <xs:keyref name="keyRef" refer="t:key">
            <xs:selector xpath="."/>
            <xs:field xpath="@ref"/>
        </xs:keyref>
    </xs:element>

    <xs:complexType name="nodeType">
        <xs:sequence>
            <xs:element name="left-child-node" type="t:nodeType" form="unqualified"/>
            <xs:element name="right-child-node" type="t:nodeType"/>
         </xs:sequence>
        <xs:attribute name="value" type="xs:integer" use="required"/>
        <xs:attribute name="id" type="xs:integer" use="required" form="qualified"/>
        <xs:attribute name="ref" type="xs:integer"/>
    </xs:complexType>
</xs:schema>
```

Kreuzen Sie an, welche Aussagen bzgl. **ns.xsd** (bzw. für ein gültiges Instanzdokument desselben) wahr bzw. falsch sind.

1. Im Instanzdokument kann für das Schlüsselattribut `id` auch ein anderer Präfix verwendet werden als der im XML Schema deklarierte Präfix `t` — wahr ⊗ falsch ◯
2. Im Instanzdokument kann das Element `left-child-node` auch einem anderen Namespace als in der XML Schema Datei zugeordnet werden. — wahr ◯ falsch ⊗
3. Das Element `left-child-node` kann im Instanzdokument im leeren Namespace liegen. — wahr ⊗ falsch ◯
4. Im XML Schema wird bereits deklariert, dass alle Attribute außer `id` keinem Namespace zugeordnet werden müssen. — wahr ⊗ falsch ◯
5. Statt dem Präfix `xs` könnte auch ein anderer Präfix im XML Schema deklariert und verwendet werden. — wahr ⊗ falsch ◯
6. Das Attribut targetNamespace kann in der XML Schema Datei auch weggelassen werden. — wahr ◯ falsch ⊗

**Aufgabe 3:** (9)

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

1. Genauso wie bei Elementen ist die Reihenfolge von Attributen innerhalb eines Elements in XML-Dokumenten signifikant. — wahr ◯ falsch ⊗
2. Wohlgeformte XML-Dokumente sind immer auch gültige XML-Dokumente. — wahr ◯ falsch ⊗
3. SAX-Parser können den Parse-Vorgang fortsetzen wenn ein Gültigkeitsfehler entdeckt wurde. — wahr ⊗ falsch ◯
4. In XPath beginnt ein absoluter Pfad immer vom Wurzelknoten aus. — wahr ⊗ falsch ◯
5. Das Typsystem von XQuery basiert auf XML Schema. — wahr ⊗ falsch ◯
6. XSLT unterstützt sowohl imperative als auch deklarative Programmiertechniken. — wahr ⊗ falsch ◯

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

**Aufgabe 4:** (12)

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

- Es soll möglich sein, dass auch keine einzige Person im XML-Instanzdokument enthalten ist.
- Entscheiden Sie selbständig anhand des XML-Instanzdokuments, ob ein Attribut in diesem definiert werden muss, oder optional ist.
- Um Integritätsbedingungen (`key`/`keyref`) müssen Sie sich nicht kümmern.

Datei **stammbaum.xsd**:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="familie">
        <xs:complexType>
            <xs:sequence>

                <xs:element name="personen">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="person" minOccurs="0" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:attribute name="id" type="xs:int" use="required"/>
                                    <xs:attribute name="name" type="xs:string" use="required"/>
                                    <xs:attribute name="gebjahr" type="xs:gYear" use="required"/>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>

                <xs:element name="stammbaum">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="person" type="personTyp"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>

            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <!-- Fortsetzung auf naechster Seite -->
```

```
    <xs:complexType name="personTyp">
        <xs:sequence>
            <xs:element name="person" type="personTyp" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="ref" type="xs:int" use="required"/>
        <xs:attribute name="verheiratet-mit" type="xs:int"/>
    </xs:complexType>
</xs:schema>
```

**Aufgabe 5:** (10)

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

- Falls als Ergebnis eine Knotenmenge selektiert wird, geben Sie als Ausgabe die Werte der `id` bzw. `ref` Attribute an.
- Falls der angegebene XPath Ausdruck keine Knoten selektiert, notieren Sie im entsprechenden Feld "leere Ausgabe".
- Falls als Ergebnis eine Zahl selektiert wird (count), geben Sie diese Zahl an.

Betrachten Sie dazu folgendes Beispiel:

```
//personen/person
```

| |
|---|
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 |

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

```
/familie/stammbaum/person/person[2]
```

| |
|---|
| 5 |

```
count(//person[@verheiratet-mit])
```

| |
|---|
| 6 |

```
//person[@ref=9]/following::*
```

| |
|---|
| 11 17 5 |

```
count(//person[@ref=3]/descendant::*)
```

| |
|---|
| 8 |

```
//person[not(following-sibling::* or preceding-sibling::*)]
```

| |
|---|
| 1 17 |

**Aufgabe 6:** (9)

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

- Zurückgegeben werden soll dieselbe Struktur wie sie das Element `<stammbaum>` aufweist.
- Anstatt des `ref` Attributs soll das Attribut `name` stehen, welches den Namen der entsprechenden Person beinhaltet.
- Das Attribut `verheiratet-mit` soll entfernt werden.

Betrachten Sie dazu folgende Ausgabe, die ihr XSLT-Stylesheet **stammbaum.xsl** angewandt auf **stammbaum.xml** (siehe Anhang) produzieren soll:

```
<stammbaum>
    <person name="George VI">
        <person name="Queen Elizabeth II">
            <person name="Prince Charles">
                <person name="Prince William of Wales"/>
                <person name="Prince Henry of Wales"/>
            </person>
            ...
        </person>
        <person name="Princes Margaret" />
    </person>
</stammbaum>
```

Vervollständigen Sie das XSLT-Stylesheet **stammbaum.xsl**. Kontrollstrukturen wie z.B. `xsl:for-each` sind für die Lösung grundsätzlich erlaubt aber nicht erforderlich (ausreichend zur Lösung des Beispiels sind wenige Templates mit jeweils relativ kurzem Inhalt). Sie brauchen sich nicht um Whitespaces etc. zu kümmern.

Datei **stammbaum.xsl**:

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

    <xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>

    <xsl:template match="/">
        <stammbaum>
            <xsl:apply-templates select="familie/stammbaum"/>
        </stammbaum>
    </xsl:template>

    <xsl:template match="person">
        <person name="{//personen//person[@id=current()/@ref]/@name}">
            <xsl:apply-templates />
        </person>
    </xsl:template>

</xsl:stylesheet>
```

**Aufgabe 7:** (9)

Betrachten Sie folgende-XQuery Abfrage **stammbaum.xq**:

```
for $person in //stammbaum//person[@verheiratet-mit][count(*) >= 2]
let $p1 := //personen/person[@id = $person/@ref]
let $p2 := //personen/person[@id = $person/@verheiratet-mit]
return
    <p>
        <p1 name="{$p1/@name}" />
        <p2 name="{$p2/@name}" />
    </p>
```

Geben Sie nun die Ausgabe von **stammbaum.xq** angewandt auf **stammbaum.xml** an.

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

```
<p>
    <p1 name="George VI"/>
    <p2 name="Queen Elizabeth"/>
</p>
<p>
    <p1 name="Queen Elizabeth II"/>
    <p2 name="Prince Philip"/>
</p>
<p>
    <p1 name="Prince Charles"/>
    <p2 name="Lady Diana"/>
</p>
<p>
    <p1 name="Prince Andrew"/>
    <p2 name="Sarah Ferguson"/>
</p>
```

**Aufgabe 8:** (8)

Vervollständigen Sie die Methode `startElement` der Klasse MySAXRefFilter, welche Dokumente der Gestalt **stammbaum.xml** als Eingabe erhält und folgende Modifikation durchführt:

Anstatt der Attribute `ref` und `verheiratet-mit` sollen die Attribute `name` und `ehepartner` stehen, welche die Namen der entsprechenden Personen beinhalten. Ansonsten soll keine Änderung am Eingabe-Dokument vorgenommen werden. Um Fehlerbehandlung müssen Sie sich nicht kümmern.

**Hinweis:** Neue Attribute können erstellt werden mittels:

```
AttributesImpl newAtts = new AttributesImpl();
newAtts.addAttribute("", "[localName]", "[qName]", "[type]", "[value]");
```

**Tipp:** Verwenden Sie die Map `persons` um die Namen der Personen beim Durchlaufen von `<personen>` zu speichern um auf diese beim Durchlaufen von `<stammbaum>` zugreifen zu können.

```
public class MySAXRefFilter extends XMLFilterImpl
{
    Map<Integer, String> persons = new HashMap<Integer, String>();

    public void startElement(String uri, String localName, String qName, Attributes atts)
        throws SAXException
    {
        if (localName.equals("person") && atts.getValue("id") != null) {
            persons.put(Integer.parseInt(atts.getValue("id")), atts.getValue("name"));
            super.startElement(uri, localName, qName, atts);
        }
        else
            if (localName.equals("person") && atts.getValue("ref") != null) {
                int refValue = Integer.parseInt(atts.getValue("ref"));
                AttributesImpl newAtts = new AttributesImpl();
                newAtts.addAttribute("", "name", "name", "xs:string", persons.get(refValue));

                if (atts.getValue("verheiratet-mit") != null) {
                    int vmValue = Integer.parseInt(atts.getValue("verheiratet-mit"));
                    newAtts.addAttribute("", "ehepartner", "ehepartner",
                        "xs:string", persons.get(vmValue));
                }

                super.startElement(uri, localName, qName, newAtts);
            }
        else
            super.startElement(uri, localName, qName, atts);
    }
}
```

**Sie können diese Seite abtrennen!**

Die Datei **stammbaum.xml** listet unter `personen` alle Familienmitglieder auf. Unter `stammbaum` wird das Verhältnis dieser Personen beschrieben.

Zum Beispiel: George VI (id=1) und Queen Elizabeth (id=2) sind verheiratet und haben zwei Kinder (Queen Elizabeth II (id=3) und Princes Margaret (id=5), welche wiederum mit anderen Personen verheiratet sind).

Datei **stammbaum.xml**:

```
<familie>

    <personen>
        <person id="1" name="George VI" gebjahr="1895"/>
        <person id="2" name="Queen Elizabeth" gebjahr="1900"/>
        <person id="3" name="Queen Elizabeth II" gebjahr="1926"/>
        <person id="4" name="Prince Philip" gebjahr="1900"/>
        <person id="5" name="Princes Margaret" gebjahr="1930"/>
        <person id="6" name="Antony Armstrong-Jones" gebjahr="1930"/>
        <person id="7" name="Prince Charles" gebjahr="1948"/>
        <person id="8" name="Lady Diana" gebjahr="1961"/>
        <person id="9" name="Prince Andrew" gebjahr="1960"/>
        <person id="10" name="Sarah Ferguson" gebjahr="1959"/>
        <person id="11" name="Prince Edward" gebjahr="1964"/>
        <person id="12" name="Sophie Rhys-Jones" gebjahr="1965"/>
        <person id="13" name="Prince William of Wales" gebjahr="1982"/>
        <person id="14" name="Prince Henry of Wales" gebjahr="1984"/>
        <person id="15" name="Princess Beatrice of York" gebjahr="1988"/>
        <person id="16" name="Princess Eugenie of York" gebjahr="1990"/>
        <person id="17" name="Louise" gebjahr="2003" />
    </personen>

    <stammbaum>
        <person ref="1" verheiratet-mit="2">
            <person ref="3" verheiratet-mit="4">
                <person ref="7" verheiratet-mit="8">
                    <person ref="13"/>
                    <person ref="14"/>
                </person>
                <person ref="9" verheiratet-mit="10">
                    <person ref="15"/>
                    <person ref="16"/>
                </person>
                <person ref="11" verheiratet-mit="12">
                    <person ref="17"/>
                </person>
            </person>
            <person ref="5" verheiratet-mit="6"/>
        </person>
    </stammbaum>

</familie>
```

Gesamtpunkte: 75