

Gruppe A	PRÜFUNG AUS "SEMISTRUKTURIERTE DATEN" 181.135		18. 01. 2012
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*)>
<!ELEMENT B (A | C)+>
<!ELEMENT C (#PCDATA | A | B)*>
<!ATTLIST A key ID #REQUIRED>
<!ATTLIST B keyref IDREF #IMPLIED>
<!ATTLIST C keyref 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. `<C>END</C>` gültig ungültig
2. `` gültig ungültig
3. `<C/><C/><C/>` gültig ungültig
4. `<C keyref="k2"/>` gültig ungültig
5. `<C>TEXT<C keyref="k2"/>TEXT</C>` gültig ungültig
6. `<C>TEXT</C><A/>` gültig ungültig

(Regeln für Beispiele 1–3: Pro korrekter Antwort 1.5 Punkte, **pro inkorrektter 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:el="http://www.dbai.tuwien.ac.at/education/ssd/SS11/pruefung/NS"
  elementFormDefault="qualified">

  <xs:element name="edgelist">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="edge" type="el:edgeType" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="edgeType">
    <xs:sequence>
      <xs:element name="point" type="el:pointtype" minOccurs="2" maxOccurs="2"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="pointtype">
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>

</xs:schema>
```

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

1. Würde man im XML Schema bei der Attributdefinition von **name** das Attribut **form** auf den Wert **qualified** setzen, dann muss im Instanzdokument bei diesem Attribut der **targetNamespace** referenziert werden. wahr falsch
2. In dem XML Schema ist es über das Attribut **targetNamespace** möglich mehrere Target-NS für Instanzdokumente zu deklarieren. wahr falsch
3. Im Instanzdokument wäre es möglich sowohl einen Präfix **e11** als auch einen weiteren Präfix **e12** zu deklarieren, welche beide auf den im XML Schema deklarierten **targetNamespace** zeigen. wahr falsch
4. Im Instanzdokument zu dem XML Schema müssen Attribute immer im leeren NS liegen. wahr falsch
5. Das Attribut **targetNamespace** dient nur der Dokumentation und kann in der XML Schema Datei auch weggelassen werden. wahr falsch
6. Im Instanzdokument zu dem XML Schema ist es möglich, dass ein Element keinem Namespace zugeordnet ist. wahr falsch

Aufgabe 3:

(9)

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

1. Gültige XML-Dokumente sind immer auch wohlgeformte XML-Dokumente. wahr falsch
2. Bei XML handelt es sich um eine Teilmenge der Meta-Markup-Sprache SGML. wahr falsch
3. Bei einer Ableitung neuer Typen in einem XML Schema mittels 'Restriction', müssen alle Komponenten, die enthalten sein sollen, noch einmal explizit angegeben werden. wahr falsch
4. DOM sieht Attribute als Kinder von Elementen an. wahr falsch
5. In XPath beginnt ein relativer Pfad immer vom aktuellen Context Node aus wahr falsch
6. XSLT ist auch ohne Kontrollstrukturen noch Turing-vollständig wahr falsch

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

Aufgabe 4:

(12)

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

- Das Element **flugplan** ist das Wurzelement und besteht aus mindestens einem **flughafen**-Elementen gefolgt von beliebig vielen **flug**-Elementen.
- Das Element **flug** besteht aus mindestens 2, aber maximal 4 **flughafen**-Elementen.
- Nicht alle der folgenden Constraints sind in einer DTD darstellbar. In so einem Fall soll zumindest sichergestellt werden, dass XML-Dokumente in der Gestalt von **flugplan.xml** als gültig erkannt werden.
 - Tritt das Element **flughafen** als Kindelement von **flugplan** auf, besteht es aus dem optionalem Element **name** gefolgt von dem Element **stadt**. Außerdem besitzt es ein Attribut **code** vom Typ **ID**.
 - Tritt das Element **flughafen** als Kindelement von **flug** auf, handelt es sich um ein leeres Element mit dem Attribut **ref** vom Typ **IDREF**.
- Attribute sollen (wenn möglich) als verpflichtend deklariert werden.
- Sollten bei bestimmten Elementen oder Attributen keine näheren Angaben bezüglich des genauen Typs vorgegeben sein, wählen Sie selbst einen sinnvollen Typ aus.

Datei **flugplan.dtd**:

Aufgabe 5:

(10)

Betrachten Sie die folgenden XPath-Abfragen angewandt auf das Dokument **flugplan.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), geben Sie diese Zahl an.

Betrachten Sie dazu folgendes Beispiel:

```
//flughafen/@code
```

```
VIE CDG ORY FCO CIA
```

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

```
/flugplan/flug[@nr = "F1"]/flughafen/@ref
```

```
//flug[@typ = "linie"][2]/@nr
```

```
//flug[2][@typ = "linie"]/@nr
```

```
count(//flug[@typ = "linie"][./flughafen/@ref = "VIE"])
```

```
//flughafen[@code = distinct-values(//flug[@typ = "charter"]/flughafen/@ref)]/stadt/text()
```

Aufgabe 6:

(9)

Erstellen Sie ein XSLT-Stylesheet **flugplan.xsl**, das angewandt auf Dokumente der Gestalt **flugplan.xml** eine Ausgabe folgender Form liefert:

```
<flightplan>
  <flight no="F1">Wien - Paris</flight>
  <flight no="G3">Wien - Paris</flight>
  <flight no="A7">Wien - Paris</flight>
  <flight no="A8">Paris - Wien</flight>
  <flight no="B4">Paris - Rom</flight>
</flightplan>
```

Beachten Sie folgende Punkte:

- Die Element- und Attributnamen haben sich gegenüber dem Ursprungsdokument geändert.
- Für jeden Flug (**flug**) soll anstatt der Subelemente **flughafen** (mit Attribut **ref**) der Name der Stadt des referenzierten Flughafens angegeben werden.
- Beachten Sie, dass grundsätzlich mehr als zwei Flughäfen pro Flug vorkommen können!

Vervollständigen Sie hier das XSLT-Stylesheet **flugplan.xsl**. Kontrollstrukturen wie z.B. **xsl:for-each** sind für die Lösung grundsätzlich erlaubt aber nicht erforderlich. Sie brauchen sich nicht um Whitespaces etc. zu kümmern (insbesondere die Bindestriche z.B. in "Wien - Paris" sind nicht unbedingt erforderlich).

Datei **flugplan.xsl**:

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

```
</xsl:stylesheet>
```

Aufgabe 7:

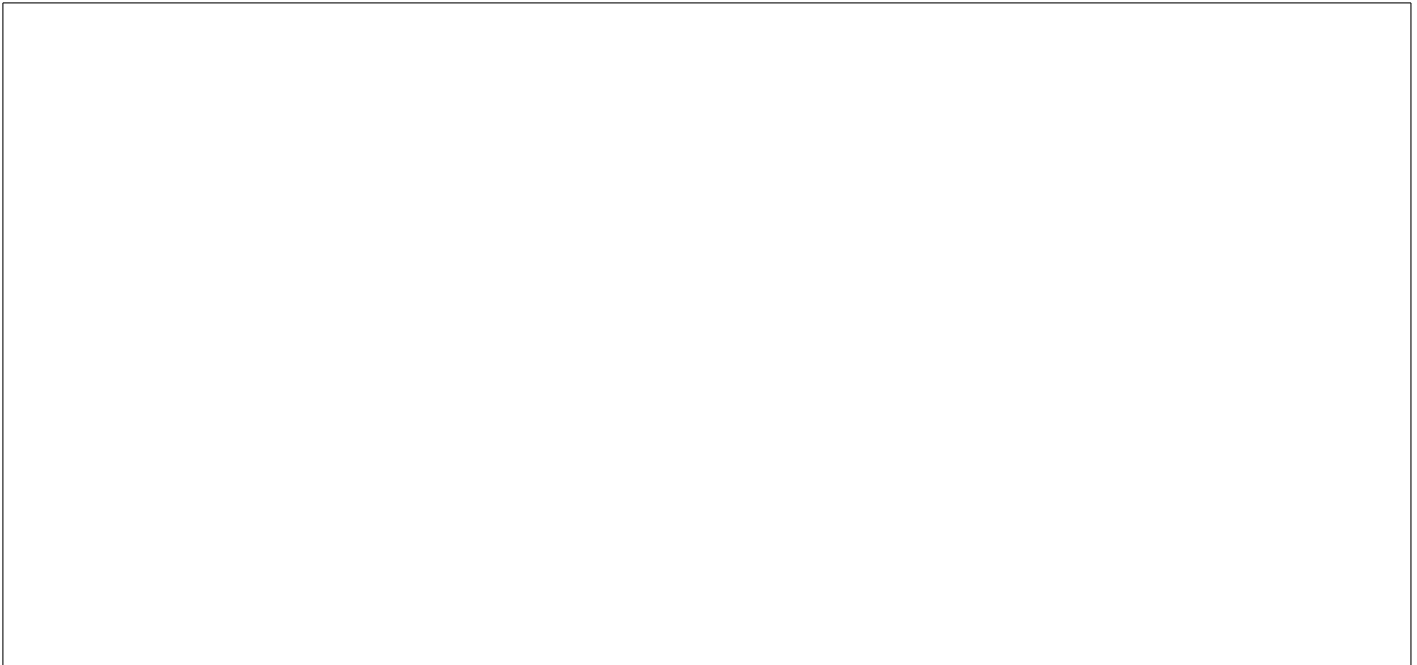
(9)

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

```
for $flughafen in doc("flugplan.xml")/flugplan/flughafen
let $fluege := count(doc("flugplan.xml")//flug[flughafen/@ref = $flughafen/@code]/@nr)
where $fluege > 2
return
<flughafen code="{ $flughafen/@code }">
<fluege>{ $fluege }</fluege>
</flughafen>
```

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

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



Aufgabe 8:

(8)

Vervollständigen Sie die Methode `findeAngeflogeneFlughafen`, die einem gegebenen DOM Element `flughafen` weitere Kindelemente `fliegt-nach` hinzufügt, welche als Textinhalt jeweils den Code von einem der angeflogenen Flughäfen enthalten.

Sie können davon ausgehen, dass der dritte Parameter `fluege` bereits genau jene `flug` Elemente enthält, welche mit dem `flughafen` Element assoziiert sind (d.h. angeflogen werden). Außerdem können Sie noch davon ausgehen, dass diese `flug` Elemente nur aus `flughafen` Kindelementen bestehen. Um Fehlerbehandlung müssen Sie sich nicht kümmern.

Für den Flughafen VIE soll das Ergebniselement dann auf das Beispieldokument bezogen folgendermaßen aussehen:

```
<flughafen code="VIE">
  <stadt>Wien</stadt>
  <fliegt-nach>CDG</fliegt-nach>
  <fliegt-nach>ORY</fliegt-nach>
</flughafen>
```

Beachten Sie dabei insbesondere, dass kein Kindelement `flight-nach` mit dem eigenen Flughafencode `VIE` hinzugefügt werden soll, und dass keine Duplikate vorkommen sollen (z.B. zweimal `CDG`).

```
public void findeAngeflogeneFlughafen(Document document, Element flughafen, List<Element> fluege)
{
```

```
}
```


Sie können diese Seite abtrennen!

Datei flugplan.xml:

```
<flugplan>
  <flughafen code="VIE">
    <stadt>Wien</stadt>
  </flughafen>
  <flughafen code="CDG">
    <name>Charles de Gaulle</name>
    <stadt>Paris</stadt>
  </flughafen>
  <flughafen code="ORY">
    <name>Orly</name>
    <stadt>Paris</stadt>
  </flughafen>
  <flughafen code="FCO">
    <name>Leonardo Da Vinci/Fiumicino</name>
    <stadt>Rom</stadt>
  </flughafen>
  <flughafen code="CIA">
    <name>Ciampino</name>
    <stadt>Rom</stadt>
  </flughafen>

  <flug nr="F1" typ="charter">
    <flughafen ref="VIE"/>
    <flughafen ref="CDG"/>
  </flug>
  <flug nr="G3" typ="linie">
    <flughafen ref="VIE"/>
    <flughafen ref="CDG"/>
  </flug>
  <flug nr="A7" typ="linie">
    <flughafen ref="VIE"/>
    <flughafen ref="ORY"/>
  </flug>
  <flug nr="A8" typ="linie">
    <flughafen ref="ORY"/>
    <flughafen ref="VIE"/>
  </flug>
  <flug nr="B4" typ="charter">
    <flughafen ref="CDG"/>
    <flughafen ref="FCO"/>
  </flug>
</flugplan>
```

Gesamtpunkte: 75