1. Versuch: Einführende XML-Aufgaben

Wichtige Informationen zu den in dieser Veranstaltung verwendeten Werkzeugen finden Sie unter
`http://ap34.ifi.informatik.uni-goettingen.de/`.

Aufgabe 1.1 (XML-Baum vs Directory-Baum)

Laden Sie `mondial-europe.xml` in *xmllint* und schauen Sie sich die Ordnerstruktur an. Wechseln Sie erst ins Land Deutschland, und dann in die Stadt Göttingen. Wechseln Sie jetzt zur übernächsten Stadt im Dokument.
Verweise auf *xmllint* und die Mondial-Dateien finden Sie unter
`http://ap34.ifi.informatik.uni-goettingen.de`.

Aufgabe 1.2

- Schreiben Sie eine DTD für XML-Dokumente mit Studentendaten:
  Ein Student hat einen Namen, eine Adresse, eine Matrikelnummer, ein oder mehrere Studienfächer usw.
- Schreiben Sie zu der DTD ein XML-Dokument, das Studentendaten enthält, und validieren Sie diese Beispielinstanz mit *xmllint*.

Aufgabe 1.3 (HTML-XHTML)

- Suchen Sie sich ein einfaches HTML-Dokument (z.B. Ihre eigene Studierenden-Homepage), und wandeln Sie sie von Hand von HTML nach XHTML um.
- Validieren Sie die Seite mit dem XHTML-Validator, der unter
  `http://ap34.ifi.informatik.uni-goettingen.de/#xhtml` angelinkt ist.

Hinweis: In Ihrem Home-Verzeichnis auf der ap34 existiert ein Verzeichnis `public_html`. Dies ist Ihr Webverzeichnis. Auf Seiten, die Sie dort ablegen, können Sie über
`http://ap34.ifi.informatik.uni-goettingen.de/`~<*username*>/<*dateiname*> per Browser zugreifen.

Aufgabe 1.4 (Programmieraufgabe)

Schreiben Sie eine Java-Klasse (oder genauer: mehrere Java-Klassen), die ein XML-Dokument als Baum repräsentiert. Folgendes ist zu beachten:

- Ein Dokument besteht aus einem Wurzelknoten, der 0 bis 5 Kinder vom Typ Element oder Text hat.
- Ein Knoten kann ein Element, ein Attribut oder ein Textknoten sein.
- Ein Elementknoten hat einen Namen (String).
- Ein Elementknoten hat 0 bis 5 Kinder (können Sie als Array der Größe 5 fest verdrahten) vom Typ Element oder Text.
- Ein Elementknoten hat 0 bis 5 Attribute.
- Ein Attributknoten hat einen Namen (String) und einen Wert (String).
- Ein Textknoten enthält Text (String).

Außerdem soll jeder Knoten eine Methode "toString()" besitzen, die in Tiefensuche durch den Baum läuft und den Inhalt des Baumes als normale, formatierte XML-ASCII-Datei ausgibt.

Aufgabe 1.5 (Abstrakter Datentyp)

Gegeben sei die Signatur des abstrakten Datentyps *Doc*, der ein XML-Dokument beschreibt.

$$
\begin{array}{lll}
create & : & \emptyset \longrightarrow Doc \\
get\_root & : & Doc \longrightarrow Node \\
get\_children & : & Node \longrightarrow List < Node > \\
get\_attributes & : & Node \longrightarrow List < Node > \\
get\_name & : & Node \longrightarrow String \\
get\_value & : & Node \longrightarrow String \\
get\_type & : & Node \longrightarrow \{\text{“attr”,”elem”,”text”}\} \\
map & : & op \times List < Node > \longrightarrow List < Node > \\
concat & : & List < String > \longrightarrow String
\end{array}
$$

- `Doc` ist der Konstruktor für das XML-Dokument
- `get_root` liefert das root-Element des Dokumentes
- `get_children` liefert die Liste aller Kindknoten (Text- und Elementknoten) eines Knotens. `get_children` von einem Attribut- oder Textknoten liefert die leere Liste.
- `get_attributes` liefert die Liste aller Attributknoten eines Knotens. `get_attributes` von einem Attribut- oder Textknoten liefert die leere Liste.
- `get_name` liefert den Namen eines Knotens als String: `get_name` von einem Elementknoten liefert den Elementnamen, `get_name` von einem Attributknoten liefert den Attributnamen. `get_name` von einem Textknoten ist leer.
- `get_value` liefert den Wert eines Knotens als String: `get_name` von einem Attributknoten liefert den Attributwert als String. `get_name` von einem Textknoten liefert den Text selbst. `get_value` von einem Elementknoten liefert die Konkatenation aller Textknoten-Nachfolgerdes Elements.
- `map` wendet eine Operation *op* auf alle Elemente einer Liste von Elementen des Typs T an und liefert die Liste der Resultate zurück.
- `concat` vereinigt eine Liste von einzelnen Strings in einen einzigen String.
- $List < T >$ steht für eine Liste von Elementen des Typs T.

Anmerkung: Eine Liste können Sie mit $[e_1, ..., e_n]$ schreiben, für eine leere Liste schreibt man [].

Verwenden Sie die Funktionen, deren Signaturen oben gegeben sind, und geben Sie Signatur und Axiome an für eine Funktion `toString` an, die einen Node nach String abbildet und diesen Node in der üblichen ASCII-Notation für XML-Dokumente darstellt. `toString(get_root(d))` schreibt also das ganze XML-Dokument *doc* als ASCII-Datei.
Hier ist die Funktionsdefinition für `toString` in Form einer Fallunterscheidung angegeben. Vervollständigen Sie die Funktionsdefinition:

$$
toString(node) = \begin{cases} get\_value(node) & \text{falls} \quad getType(node) = \text{“text”} \\ ... & \text{falls} \quad getType(node) = \text{“attr”} \\ ... & \text{falls} \quad ... \end{cases}
$$

Aufgabe 1.6 (Endliche Automaten und DTDs) Gegeben sei folgende DTD:

```
<!ELEMENT date (day,month,year?)>
<!ELEMENT day (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT a (date*)>
<!ELEMENT b (#PDCATA)>
<!ELEMENT c (b+|(b?,a)*)>
```

Geben Sie zu jeder Elementdefinition einen deterministischen endlichen Automaten an, der das jeweilige Content Model akzeptiert.
Hinweis: Starten Sie mit den Automaten für die einfachen Content Models und bauen Sie die komplizierteren Automaten aus den einfachen zusammen.