Vorlesung XML im Wintersemester 2003/2004
Test XSLT, 27.1.2004, 13^{15} - 14^{15}

MUSTERLÖSUNG

Name: _____

Da XSLT Code durch die XML-Syntax schnell unübersichtlich wird, können Sie in diesem Test eine Pseudo-Notation verwenden, in der Sie sich sie End Tags sparen und Schachtelungen statt dessen durch Klammern markieren. Sie können auch Start Tags vereinfachen (kein Namespace Prefix, keine spitzen Klammern), aber die Syntax der Attribute sollte erhalten bleiben. Anweisungen werden durch Semikolon getrennt. Hier ein kleines Beispiel:

`<xsl:for-each select="test">`
`  <xsl:value-of select="child"/>`
`  <xsl:text>  </xsl:text>`
`</xsl:for-each>`
	`for-each select="test" {`
	`  value-of select="child" ;`
	`  text { "  " } ;`
	`}`

Sollte Ihnen der Pseudocode nicht gefallen, so können Sie natürlich auch kompletten XSLT Code schreiben, bei reinen Syntaxfehlern gibt es keine Abzüge in der Bewertung!

1) XSLT kennt keine Iterationen über Variablen (sondern nur über Node Sets mit der `for-each` Anweisung), so dass man Iterationen über Zahlenbereiche rekursiv lösen muss. Dazu verwendet man i.A. *Named Templates* mit *Parametern*. Schreiben Sie den folgenden iterativen Pseudocode in rekursiven Pseudocode um (denken Sie dabei daran, dass Sie in XSLT den Wert von Variablen *nicht verändern* können):

```
FUNCTION A (X, Y) {
      FOR I = 1 TO X {
            FOR J = 1 TO Y {
                  PRINT('I = ",I,"; J = ",J,";") } } }
```

   Als Ziel Ihrer Lösung sollen die das folgende XSLT Named Template ausprogrammieren, hier als Gerüst in Pseudocode vorgegeben:

```
template name="A" {
      param "X" ; param "Y" ;
      if $X >= 1 { call-template A (X = $X-1; Y = $Y) };
      call-template B (X = $X; Y = $Y);
      }

template name="B" {
      param "X" ; param "Y" ;
      if $Y > 1 {call-template B (X = $X; Y = $Y-1) };
      value-of(concat("I = ",$X,";J = ",$Y));
      }
```

2) Man kann direkt in XPath recht viele Probleme lösen, die man auch in XSLT Code programmieren könnte. Beschreiben Sie ein Beispiel Ihrer Wahl, bei dem man eine Aufgabe sowohl in einem einzigen XPath (z.B. durch Prädikate), als auch in XSLT Code (z.B. mit Iterationen und Fallunterschiedungen) lösen kann. Schreiben Sie beide

Codeteile auf (sie müssen funktional *identisch* sein!) und beschreiben Sie danach Vor- und Nachteile beider Ansätze.

XPath: `for-each "//p[ancestor::table][descendant::*]" { … }`
XSLT:
```
for-each "//p"
          { if "ancestor::table" {
                if "descendant::* { … } } }
```
Vorteil der XPath-basierten Lösung ist, dass sie kompakter ist. Zudem könnte eine gute Implementierung eines XSLT-Prozessors auf Grund der deklarativen Natur des Ausdrucks einfacher Optimierungen vornehmen. Nachteil der XPath-Lösung ist, dass sie u.U. etwas mehr Überlegungen erfordert als das Ausprogrammieren.
Die Code-basierte Lösung ist für XSLT-Einsteiger einfacher zu schreiben und zu verstehen, sie ist jedoch fehleranfälliger und u.U. ineffizienter.

3) Schreiben Sie ein allgemeingültiges XSLT Programm, das ein beliebiges XML Dokument als Eingabe nimmt und nahezu unverändert wieder ausgibt, mit der Ausnahme, dass Elemente, die (in XML Schema gesprochen) "Simple Type Werte" enthalten, auf Attribute abgebildet werden. Berücksichtigen Sie, dass mehrere solcher Elemente innerhalb eines Elementes vorkommen können und Sie deshalb auf Namenskonflikte achten müssen. Lösen Sie diese Namenskonflikte, indem sie die Elemente numerieren (aus `<name>Wilde</name>` wird `name-1="Wilde"`)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:strip-space elements="*"/>
 <xsl:template match="/ | node() | @*">
  <xsl:copy>
   <xsl:apply-templates select="@*"/>
   <xsl:apply-templates select="node()"/>
  </xsl:copy>
 </xsl:template>
 <xsl:template match="*[not(@*)][count(node()) = 1][text()]">
  <xsl:attribute name="{local-name()}-{position()}">
   <xsl:value-of select="."/>
  </xsl:attribute>
 </xsl:template>
</xsl:stylesheet>
```