

Examen de PXML

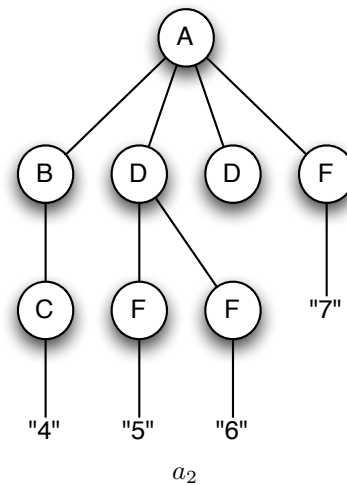
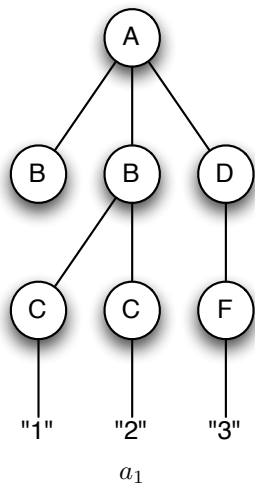
vendredi 1 avril 2011

durée 3h

supports de cours autorisés

Exercice 1 :

On considère les deux arbres XML suivants a_1 et a_2 :



Question 1 : Traduire ces arbres en documents XML de noms respectifs $a_1.xml$ et $a_2.xml$.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE A SYSTEM "a3.dtd">
<A>
  <B/>
  <B>
    <C>1</C>
    <C>2</C>
  </B>
  <D>
    <F>3</F>
  </D>
</A>
```

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE A SYSTEM "a3.dtd">
<A>
  <B>
    <C>4</C>
  </B>
  <D>
    <F>5</F>
    <F>6</F>
  </D>
</D/>
<F>7</F>
</A>

```

Question 2 : Définir une DTD qui valide le document a1.xml mais pas le document a2.xml.

```

<!ELEMENT A (B+,D+ )>
<!ELEMENT B (C*)>
<!ELEMENT C (#PCDATA)>
<!ELEMENT D (F*)>
<!ELEMENT F (#PCDATA)>

```

Question 3 : Définir une DTD qui valide le document a2.xml mais pas le document a1.xml.

```

<!ELEMENT A (B+,D+,F)>
<!ELEMENT B (C*)>
<!ELEMENT C (#PCDATA)>
<!ELEMENT D (F*)>
<!ELEMENT F (#PCDATA)>

```

Question 4 : Définir une DTD qui valide les deux documents a1.xml et a2.xml.

```

<!ELEMENT A (B+,D+,F?)>
<!ELEMENT B (C*)>
<!ELEMENT C (#PCDATA)>
<!ELEMENT D (F*)>
<!ELEMENT F (#PCDATA)>

```

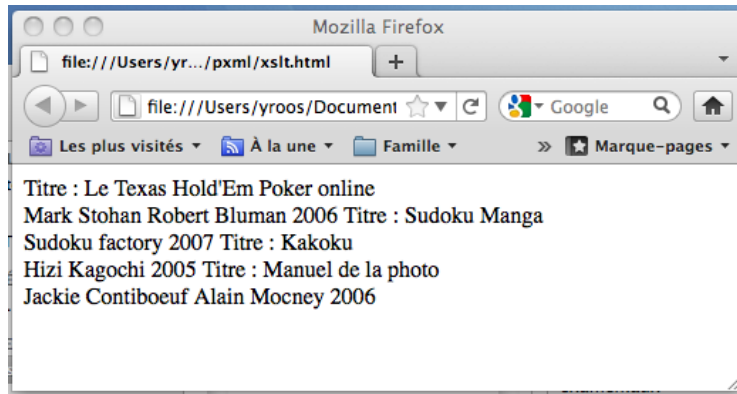
Exercice 2 : On considère le fichier XML, de nom `liste.xml`, ci-dessous :

```
<?xml version="1.0" encoding="UTF-8"?>
<liste>
  <livre>
    <titre genre="jeu">Le Texas Hold'Em Poker online</titre>
    <auteur>Mark Stohan</auteur>
    <auteur>Robert Bluman</auteur>
    <parution>2006</parution>
  </livre>
  <livre>
    <titre genre="jeu">Sudoku Manga</titre>
    <auteur>Sudoku factory</auteur>
    <parution>2007</parution>
  </livre>
  <livre>
    <titre genre="jeu">Kakoku</titre>
    <auteur>Hizi Kagochi</auteur>
    <parution>2005</parution>
  </livre>
  <livre>
    <titre genre="photo">Manuel de la photo</titre>
    <auteur>Jackie Contiboef</auteur>
    <auteur>Alain Mocney</auteur>
    <parution>2006</parution>
  </livre>
</liste>
```

On considère également la feuille XSLT suivante :

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes"/>
  <xsl:template match="/">
    <html>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="titre">
    Titre : <xsl:value-of select="."/>
    <br/>
  </xsl:template>
</xsl:stylesheet>
```

Question 1 : La visualisation du fichier HTML produit par cette feuille de style correspond elle à la capture d'écran ci-dessous ? Expliquer pourquoi.



Oui car l'`apply-templates` est lancé sur tous les éléments. Pour `titre`, la règle existe et est appliquée. Pour les autres la règle par défaut s'exécute en extrayant le texte de ces éléments.

On considère maintenant l'autre feuille de style suivante :

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes"/>
  <xsl:template match="/liste">
    <html>
      <body>
        <table>
          <xsl:for-each select="livre">
            <tr>
              <td>
                <xsl:value-of select="titre"/>
              </td>
              <td>
                <xsl:value-of select="parution"/>
              </td>
              <xsl:for-each select="auteur">
                <td>
                  <xsl:value-of select="."/>
                </td>
              </xsl:for-each>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Question 2 : Expliquer¹ ce que produit cette feuille XSLT quand elle est appliquée sur le fichier

1. En français, pas de code !

liste.xml.

Elle produit une table HTML contenant une ligne par livre contenant séquentiellement une cellule pour le titre, une cellule pour l'année et une cellule pour chaque auteur.

Question 3 : Donner une feuille XSLT équivalente (c'est à dire qui produit la même transformation) sans aucune instruction `xsl:for-each`.

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes" />
  <xsl:template match="/liste">
    <html>
      <body>
        <table>
          <xsl:apply-templates select="livre" />
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="livre">
    <tr>
      <xsl:apply-templates select="titre" />
      <xsl:apply-templates select="parution" />
      <xsl:apply-templates select="auteur" />
    </tr>
  </xsl:template>
  <xsl:template match="titre | parution | auteur">
    <td>
      <xsl:value-of select="." />
    </td>
  </xsl:template>
</xsl:stylesheet>
```

Exercice 3 : On considère le fichier XML ci-dessous qui représente le stock d'un maraîcher :

```
<?xml version="1.0" encoding="UTF-8"?>
<produits>
  <fruit type="clementine" prix="290" calibre="1">
    <producteur>Production Bastia</producteur>
    <origine region="Corse">France</origine>
    <qtity>15</qtity>
    <note>Sans pepins , avec feuilles</note>
    <bio />
  </fruit>
  <legume type="courgette" prix="300" calibre="2">
    <producteur>Madrid Hortelano</producteur>
    <origine>Espagne</origine>
    <qtity>100</qtity>
    <bio />
  </legume>
  <legume type="chou fleur" prix="090" calibre="2">
    <producteur>Pontivy et Cie</producteur>
    <origine region="Bretagne">France</origine>
    <qtity>100</qtity>
  </legume>
```

```

<legume type="salade" prix="075" calibre="3">
  <producteur>Marius Production</producteur>
  <origine region="Provence">France</origine>
  <qty>35</qty>
  <note>Batavia</note>
</legume>
<fruit type="melon" prix="150" calibre="1">
  <producteur>Marius Production</producteur>
  <origine region="Provence">France</origine>
  <qty>50</qty>
  <note>Melon brode</note>
  <bio/>
</fruit>
</produits>

```

Question 1 : Donnez des requêtes XPath pour sélectionner les éléments suivants (ces requêtes doivent bien sûr fonctionner sur tout document de même nature que celui de l'exemple.)

1. les producteurs de fruits.
//fruit/producteur
2. les légumes produits en Espagne.
//legume[origine="Espagne"] ou //legume[origine="Espagne"]/@type
3. les origines des clémentines de calibre 1 issues de l'agriculture biologique.
//fruit[@type="clementine" and @calibre="1" and bio]
4. les producteurs bretons.
//producteur[../origine/@region="Bretagne"]

Question 2 : On souhaite réorganiser le fichier XML pour placer d'abord les légumes, puis les fruits. L'ensemble des articles est encore placé dans un élément `produits`. Proposez une requête XQuery ou une transformation XSLT, au choix, réalisant cette transformation.

XQuery

```

xquery version "1.0";
<produits>
{
  doc("produits.xml")//legume ,
  doc("produits.xml")//fruit
}
</produits>

```

XSLT

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/">
    <produits>
      <xsl:copy-of select="//legume"/>
      <xsl:copy-of select="//fruit"/>
    </produits>
  </xsl:template>
</xsl:stylesheet>

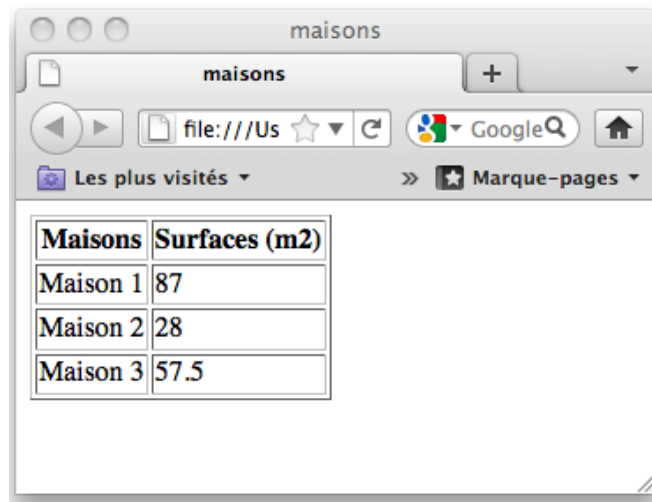
```

Exercice 4 :

Voici de nouveau le fichier de description de logements que vous connaissez-bien :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<maisons>
  <maison id="1">
    <RDC>
      <cuisine surface-m2="12">Evier Inox. Mobilier encastré</cuisine>
      <WC>Lavabo.</WC>
      <sejour>Cheminee en pierre. Baie vitree</sejour>
      <bureau surface-m2="14">Bibliotheque</bureau>
      <garage/>
    </RDC>
    <etage>
      <terrasse/>
      <chambre surface-m2="28" fenetre="3">
        <alcove surface-m2="8"/>
      </chambre>
      <chambre surface-m2="18"/>
      <sallededeBain surface-m2="15">
        Douche, baignoire, lavabo
      </sallededeBain>
    </etage>
  </maison>
  <maison id="2">
    <RDC>
      <cuisine surface-m2="12">en ruine</cuisine>
      <garage/>
    </RDC>
    <etage>
      <mirador surface-m2="1">
        Vue sur la mer
      </mirador>
      <sallededeBain surface-m2="15">Douche</sallededeBain>
    </etage>
  </maison>
  <maison id="3">
    <RDC>
      <sejour surface-m2="40"/>
    </RDC>
    <etage>
      <chambre surface-m2="17.5">Exposition plein sud</chambre>
    </etage>
  </maison>
</maisons>
```

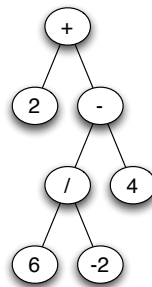
Question 1 : Écrire un programme XQuery qui, à partir de ce fichier XML, calcule, pour chaque maison, sa superficie totale. La sortie du programme sera un fichier HTML dont la visualisation correspond à la capture écran suivante :



Maisons	Surfaces (m2)
Maison 1	87
Maison 2	28
Maison 3	57.5

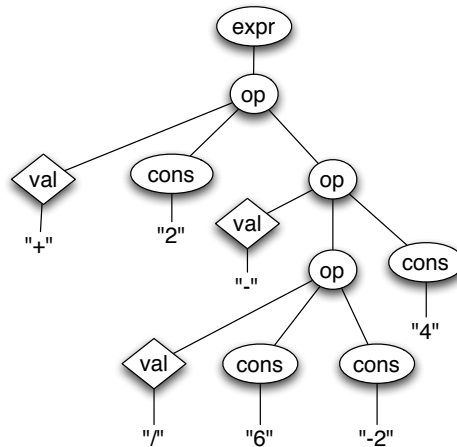
```
xquery version "1.0";
<html>
<head><title>maisons</title></head><body>
<table border="1">
  <tr><th>Maisons</th><th>Surfaces (m2)</th></tr>
  {
    for $maison in doc("maisons.xml")//maison
    return
    <tr>
      <td>Maison {string($maison/@id)}</td>
      <td>{sum($maison/*/*/@surface-m2)}</td>
    </tr>
  }
</table>
</body>
</html>
```

Exercice 5 : On souhaite stocker dans des fichiers XML des expressions arithmétiques entières en utilisant leur représentation arborescente. Si on considère, par exemple, l'expression arithmétique $(2 + ((6 / -2) - 4))$, sa représentation arborescente est :



On décide de représenter une expression par un élément racine de nom **expr** qui ne possède qu'un seul élément fils qui peut être une constante ou un opérateur. On représente les constantes entières par des éléments de nom **cons** dont le contenu est la valeur entière correspondante, et les opérateurs par des éléments de nom **op** possédant un attribut de nom **val** dont la valeur peut être "+", "-", "*" ou "/". Chaque élément de nom **op** a exactement deux éléments fils (correspondant aux sous-expressions gauche et droite), chacun d'entre eux étant donc soit un élément **cons**, soit

un élément `op`. Ainsi, l'expression précédente peut être représentée par l'arbre XML :



Ce qui donne au bout du compte le fichier XML suivant :

```

<?xml version="1.0" encoding="utf-8"?>
<expr xsi:noNamespaceSchemaLocation="expression.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <op value="+">
    <cons>2</cons>
    <op value="-">
      <op value="/">
        <cons>6</cons>
        <cons>2</cons>
      </op>
      <cons>4</cons>
    </op>
  </op>
</expression>
  
```

Question 1 : Les documents du type du fichier précédent doivent être validés par un schéma de nom `expression.xsd` comme on le voit dans l'élément racine du document. Définir un XML Schema correspondant à `expression.xsd`

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:simpleType name="type-operation">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="+"/>
      <xsd:enumeration value="-"/>
      <xsd:enumeration value="*"/>
      <xsd:enumeration value="/">
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name="cons" type="xsd:integer"/>

  <xsd:element name="op">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice>
          <xsd:element ref="cons"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
  
```

```

        <xsd:element ref="op" />
    </xsd:choice>
    <xsd:choice>
        <xsd:element ref="cons" />
        <xsd:element ref="op" />
    </xsd:choice>
</xsd:sequence>
<xsd:attribute name="val" type="type-operation" />
</xsd:complexType>
</xsd:element>

<xsd:element name="expr">
    <xsd:complexType>
        <xsd:choice>
            <xsd:element ref="cons" />
            <xsd:element ref="op" />
        </xsd:choice>
    </xsd:complexType>
</xsd:element>

</xsd:schema>

```

Dans la suite, on dira qu'un fichier XML est de type **expression** si celui-ci décrit une expression arithmétique, c'est à dire si le fichier est validé par le schéma **expression.xsd**.

Question 2 : Définir une transformation XSLT qui transforme tout fichier XML de type **expression** en un texte correspondant à l'écriture infixe complètement parenthésée de l'expression arithmétique correspondante. Par exemple, cette transformation appliquée au fichier XML de l'exemple doit sortir le texte $(2 + ((6 / -2) - 4))$

```

<?xml version="1.0" encoding="UTF-8" ?>

<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

  <xsl:output method="text" indent="yes" />

  <xsl:template match="/">

    <xsl:apply-templates select="./expression/*" />
  </xsl:template>

  <xsl:template match="op">
    (
      <xsl:apply-templates select="./*[1]" />
      <xsl:value-of select="./@val" />
      <xsl:apply-templates select="./*[2]" />
    )
  </xsl:template>

  <xsl:template match="cons">
    <xsl:value-of select="." />
  </xsl:template>

</xsl:stylesheet>

```

Question 3 : Définir un programme XQuery qui évalue l'expression arithmétique décrite dans un fichier XML de type `expression`. Par exemple, ce programme appliqué au fichier XML de l'exemple doit sortir le texte `<?xml version="1.0" encoding="UTF-8" ?>-5`

```
xquery version "1.0";

declare function local:eval($value as element()) as xs:integer
{
  let $name := node-name($value) cast as xs:string
  return
    if ($name="cons") then $value cast as xs:integer
    else
      let $v := $value/@val
      let $o1 := local:eval($value/*[1])
      let $o2 := local:eval($value/*[2])
      return
        if ($v="+") then $o1 + $o2
        else if ($v="-") then $o1 - $o2
        else if ($v="*") then $o1 * $o2
        else if ($v="/") then $o1 idiv $o2
        else 0
} ;

local:eval(/expr/*)
```