2. Unit: Querying with XPath

Solve the following exercises using XPath only.

Exercise 2.1 (XPath: Mondial)

a) Find out which countries are neighbors of Russia and have more than 10 million inhabitants.

```
//country[border/id(@country)/name='Russia' and
          population > 10000000]/name

//country[name='Russia']/border/id(@country)[population>10000000]/name
```

b) Which countries are members of the NATO? Return the countries' names.

```
//organization[abbrev="NATO"]/members/id(@country)/name/text()

//country[id(@memberships)/abbrev="NATO"]/name/text()
```

c) Give the names of countries with a neighbor country with a mountain of 4000 m and higher.

```
//mountain[elevation>=4000]/id(@country)/border/id(@country)/name

Or using the ‘‘backward’’ idref function:
//country[border/idref(@country)/parent::mountain[elevation>=4000]]/name

(: 102 hits :)
```

Exercise 2.2 (XPath: Hamlet)

a) List all scenes with less than 10 persons speaking by their titles.

```
//SCENE[count(distinct-values(.//SPEAKER))<10]/TITLE
```

b) Give the names of all persons speaking in both the first and the last act.
Give the names of all persons speaking in the first act, but *not* speaking in the last act?

```
first try:
//ACT[1]//SPEAKER[. = //ACT[position()=last()]//SPEAKER]
contains duplicates.
Thus, apply
distinct-values(//ACT[1]//SPEAKER[. = //ACT[position()=last()]//SPEAKER])
(Horatio, King Claudius, Laertes, Hamlet, Queen Gertrude, All)
speakers in the first, and not in the last:
distinct-values(//ACT[1]//SPEAKER[not(. = //ACT[position()=last()]//SPEAKER)])
(note the existential semantics of the inner comparison ‘‘not equal
to any SPEAKER’’)
(Bernardo, Francisco, Marcellus, Cornelius, Voltimand, Lord Polonius, Ophelia,
Ghost)
```

c) What happens (stage directive) directly before King Claudius says: “Part them; they are incensed.”?

```
(: note: preceding-sibling is a backward axis! :)
//SPEECH[SPEAKER="KING CLAUDIUS" and LINE="Part them; they are incensed."]
  /preceding-sibling::STAGEDIR[1]
```

```
(: <STAGEDIR>LAERTES wounds HAMLET; then in scuffling, they
   change rapiers, and HAMLET wounds LAERTES</STAGEDIR> :)
```

d) Who says what in the 187th speech overall?
The other way round (and harder): Give the query with which one can find out that the SPEECH where "Something is rotten in the state of Denmark" is said, is the 187th.

```
(: Note the difference between //SPEECH and /descendant::SPEECH
   //SPEECH[187] returns nothing since it would return all SPEECHes
   that are the 187th child of their parent :)
/descendant::SPEECH[position()=187]
(:  Marcellus: Something is rotten in the state of Denmark. :)
/descendant::SPEECH[contains(.,"Something is rotten")]/count(preceding::SPEECH)
186
```

Exercise 2.3 (XPath: Mondial (2))

a) Which (country) capitals are located at a river, sea or lake? Give their names.

```
//country/id(@capital)[located_at/@watertype]/name
```

b) Give the names of all capitals located at a lake.

```
//country/id(@capital)[located_at/@watertype="lake"]/name
```

c) Give the names of all lakes with no city located at them.

```
//lake[not (@id = //city/located_at/@lake)]/name
```

d) Give the names of all rivers flowing through some capital.

```
//country/id(@capital)/located_at/id(@river)/name (: 14 hits, only country capitals :)
id(//@capital)/located_at/id(@river)/name (: 33 hits, country and province capitals :)
```

e) Find all "german leaf-nodes", which means all element nodes that are sub-nodes of the country-element of Germany and have no children.

```
//country[name="Germany"]//*[count(./*) = 0]
//country[name="Germany"]//*[not(./*)]
```

f) In Mondial, there exist city elements as sub-elements of province elements, and city elements as sub-elements of country elements. Are there any other city elements?

```
(: returns nur country- und province-Elemente. :)
/mondial//*[(./city)]/name()
Or:
 //*[./city][(name() != province) and (name() != country)]
returns nothing.
```

Exercise 2.4 (XPath: Mondial (3))
These are some really hard examples for pure XPath.
Try to solve them with XPath, otherwise use XQuery.

a) Which organizations have at least one member on each continent? Give their names.

```
//organization[
  every $cont in //continent
```

```
  satisfies
    (some $c in //country
     satisfies
     (some $cont2 in $c/encompassed/id(@continent) satisfies $cont2 is $cont)
     and
     (some $memberIn in $c/id(@memberships) satisfies $memberIn is .))]
  /name
(: note:
   * the final "." is "self" of the organization,
   * the inner "some"s are only necessary since "is" does not accept sequences as
     arguments.
:)
```

b) Give the names of all mountains that are the highest ones on the continent where they are located.

```
//mountain[
  some $cont in ./id(@country)/encompassed/id(@continent)
  satisfies
  ( (: all other countries where mt is located are also on this continent,
       excludes Pik Chan Tengri (KAZ, KGZ, CN) from Europe!) :)
    ( every $country in ./id(@country)
      satisfies (some $cont2 in $country/encompassed/id(@continent)
                 satisfies $cont2 is $cont) )
    and
    (: no other mountain on this continent is higher :)
    not (some $mt2 in
         //mountain[every $country in ./id(@country)
                    satisfies (some $cont3 in $country/encompassed/id(@continent)
                               satisfies $cont3 is $cont)]
         satisfies number($mt2/elevation) > ./elevation))]/name
(: note:
   * the final "." is "self" of the mountain,
   * the inner "some"s for $cont2 and $cont3 is only necessary since "is"
     does not accept sequences as arguments.
:)
```

```
(: Using XQuery, it is much easier: :)
for $cont in //continent
let $mts :=
  //mountain[some $cont2 in id(@country)/encompassed/id(@continent)
             satisfies $cont2 is $cont],
    $maxelev := max($mts/elevation),
    $maxmt := $mts[elevation = $maxelev]
return
<continent name="{$cont/name}'' mountain=''{$maxmt/name}" elev="{$maxelev}''/>
```

Or, use XQuery 3.0's group by. It does not make it easier:

```
 for $m in //mountain
 group by $cont:= $m/id(@country)/encompassed/id(@continent)/name
```

does not work with mountains that are located in a country that is encompassed by more than one continent. So, for each combination (mountain, continent), a pair must be created before:

```
for $mtcont in
  (for $m in //mountain,
      $cont in $m/id(@country)/encompassed/id(@continent)/name
   return <m cont="{$cont}"> {$m/@*} {$m/*} </m> )
group by $cont:= $mtcont/@cont
let $maxelev := max($mtcont/elevation),
    $maxmt := $mtcont[elevation = $maxelev]
return
<continent name="{$cont}" mountain="{$maxmt/name}" elev="{$maxelev}"/>
```

Exercise 2.5 (XML Tree and XPath Axes)
Consider the XPath axes in a document. Provide equivalent characterizations of the "following" axis and of the "preceding" axis

i) in terms of "preorder" and "postorder" (i.e., enumeration in a preorder/postorder tree traversal),

ii) in terms of other axes.

Given an XML document, let $pre : element \cup text \to \mathbb{N}$ and $post : element \cup text \to \mathbb{N}$ denote the preorder and postorder numberings.

- Given a node x, $preceding(x)$ are all elements and text nodes y such that $pre(y) < pre(x)$ and not $post(y) < post(x)$ (this excludes the ancestors). Enumerate them in reverse document order ("preceding" is a backward axis).
- Symmetric: Given a node x, $following(x)$ are all elements and text nodes y such that $post(y) > post(x)$ (note that this does not include the descendants, since the root of a subtree is visited after the tree) and not $pre(y) < pre(x)$ (this excludes the ancestors). Enumerate them in document order.
- The nodes on the "following" axis can be enumerated as "descendants-or-self of all following siblings of the ancestors-or-self of x".
  XPath Expression:
  doc('mondial.xml')//city[name='Karlsruhe']/ancestor-or-self::*/following-sibling::*/descendant-or-self::*/name()
  Analogously for the "preceding" axis.