# CompSci 316 Fall 2015: Homework #3

***100 points (8.75% of course grade) + 10 points extra credit***
***Assigned: Thursday, October 15***
***Due: Tuesday, November 3***

This homework should be done in parts as soon as relevant topics are covered in lectures. If you wait until the last minute, you might be overwhelmed.

For Problem 1, you will need to use Gradiance. Access Gradiance via the "Gradiance" link on the course website. There is no need to turn in anything else for these problems; your scores will be tracked automatically. For other problems, you will need to turn in the required files through WebSubmit; there is a shortcut to it on the menu bar across the top of the course website. When submitting your work, make sure you select the correct course and homework. Multiple submissions are okay, but please upload ***all*** required files in each resubmission.

Problems 2, 3, and X1 should be completed on your course VM. Before you start, make sure you refresh your VM, by logging into your VM and issuing the following command:

```
/opt/dbcourse/sync.sh
```

While you may develop and debug your program for Problems 3 and X1 on your own computer, make sure it is possible to compile and run your program on the course VM, because we will use the VM for grading.

## Problem 1 (15 points)

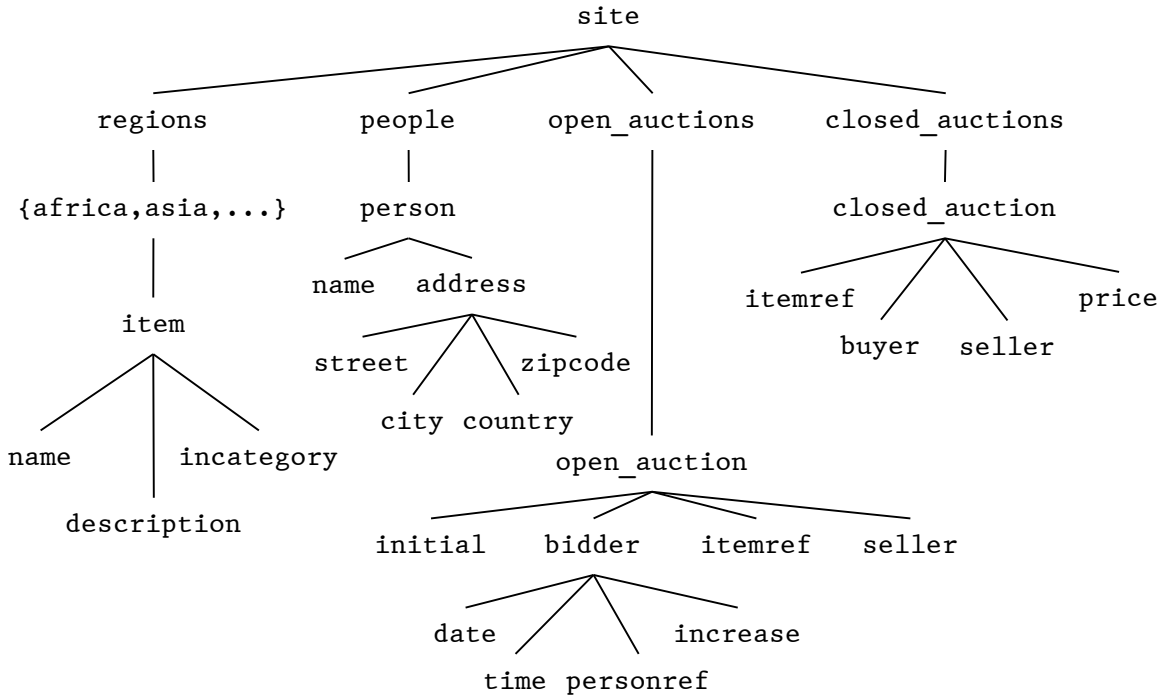
Complete the Gradiance homework titled "Homework 3.1 (XML)."

## Problem 2 (55 points)

Consider an XML document (from a project called *XMark*) modeling the data maintained by an Internet auction site in `/opt/dbcourse/examples/xmark/auction.xml`. The main entities of interest are: items, persons, open auctions, closed auctions, and categories.

- Item elements describe items that are for sale or already have been sold. Each item carries a unique identifier and bears properties like payment method (credit card, money order, etc.), a reference to the seller, a description, etc., all encoded as subelements. Each item belongs to a world region represented by the item's parent.
- Open auctions are auctions in progress. Their properties include the bid history (i.e., increases over time) along with references to the bidders, a reference to the seller, a reference to the item being sold, etc.
- Closed auctions are auctions that are finished. Their properties include references to the seller and the buyer, a reference to the respective item, the price, the quantity of items sold, etc.
- Persons are characterized by name, email address, phone number, mail address, profile of their interests, a set of open auctions they watch, etc.
- Categories feature a name and a description; they are used to implement classification of items. A category graph links categories into a network.

The figure below roughly illustrates part of the document structure that may be relevant to the problem. For the complete structure, refer to the file `auction.dtd`.




Please refer to the document "XML Tips" on the course Web site for instructions on running `saxonb-xquery`, the Saxon XQuery processor. Write queries in XQuery to answer the following questions. Because Saxon does not use any indexes and does not have a sophisticated optimizer, query performance may be heavily influenced by the way you write your queries. If a particular query takes forever to run, consider reordering loops and evaluating selections (filters) as early as possible. Note that you can add comments to your queries by enclosing them in "`(:`" and "`:)`".

For each question below, say (a), write your XQuery in a file named `2a.xq`, and generate the output file `2a.xml` by running

```
saxonb-xquery -s /opt/dbcourse/examples/xmark/auction.xml 2a.xq > 2a.xml
```

Turn in all your `.xq` and output `.xml` files.

(a) Find names of all items in "`africa`" region.

(b) Find names of all items that belong to "`category2`".

(c) Find names of all persons whose address has zip code `27`.

(d) Find all buyers who paid less than $10 in a closed auction.

(e) Find names of all persons who have bidden in an open auction for an item whose name contains the string "`cow`".

(f) Find names of all persons with address in "`United States`" who never bought anything in closed auctions.

(g) For each open auction whose seller's name is "`Venkatavasu Takano`", print out the following information:

```
<open_auction id="...">
  <bidders total_number="...">
    <bidder_name>...</bidder_name>
    <bidder_name>...</bidder_name> ...
  </bidders>
</open_auction>
```

## Problem 3 (30 points)

In `/opt/dbcourse/assignments/hw3/` on your VM, you will find an XML file `congress.xml` containing information about the current (114th) US Congress. Logically, the file consists of two sections:

- Each `person` element under `congress/people` stores information about a legislator, including the role he or she serves in the Congress. The person is a current member of the House (or Senate) if he or she has a `role` with type "`rep`" (or "`sen`", respectively) and `current` equal to 1.
- Each `committee` element under `congress/committees` stores information about a committee. It has a list of members, whose ids reference those of `person` elements in the first section; `role` specifies the role of the member in the committee (e.g., chair or ranking member). Oftentimes a committee can be divided into subcommittees. Each `subcommittee` element has its own list of members, which should be a subset of the committee members. A legislator can serve on multiple committees, and even multiple subcommittees under the same committee.

Your job is to produce an output XML file `percom.xml`, which presents information about legislators and their committee assignments in a more concise and readable form. The output file should be structured as follows, and conform to the DTD in `/opt/dbcourse/assignments/hw3/percom.dtd`.

- The root element is `congress`.
- `congress` has two child elements: `house` and `senate`, each listing its current legislators. See the description of `congress.xml` above for how to determine who are current members of the two chambers.
- Each legislator is represented as a `person` element, with a name attribute whose value is taken from `person/@name` in `congress.xml`. Under `person`, list each committee that this legislator serves in as a `committee` element. A `committee` element has a `name` attribute whose value is taken from `committee/@displayname` in `congress.xml`; it also has a `role` attribute whose value is taken from `member/@role` (or simply "`Member`" if no role is specified). Under `committee`, list each subcommittee of the committee that this legislator serves in, as a `subcommittee` element. Like a `committee` element, a `subcommittee` has a `name` attribute and a `role` attribute.

For example, here is a snippet of the output showing the committee assignment for Sen. Marco Rubio from Florida:

```
<?xml version="1.0" encoding="UTF-8"?>
<congress>
   <house>
      …
   </house>
   <senate>
      …
      <person name="Marco Rubio">
         <committee name="Senate Select Committee on Intelligence" role="Member"/>
         <committee name="Senate Committee on Commerce, Science, and Transportation" role="Member">
            <subcommittee name="Communications, Technology, Innovation, and the Internet"
role="Member"/>
            <subcommittee name="Oceans, Atmosphere, Fisheries, and Coast Guard" role="Chairman"/>
            <subcommittee name="Space, Science, and Competitiveness" role="Member"/>
            <subcommittee name="Aviation Operations, Safety, and Security" role="Member"/>
         </committee>
         <committee name="Senate Committee on Foreign Relations" role="Member">
            <subcommittee name="Africa and Global Health Policy" role="Member"/>
            <subcommittee name="East Asia, the Pacific, and International Cybersecurity Policy"
role="Member"/>
            <subcommittee name="Near East, South Asia, Central Asia, and Counterterrorism"
role="Member"/>
            <subcommittee name="Western Hemisphere, Transnational Crime, Civilian Security, Democracy,
Human Rights, and Global Women's Issues" role="Chairman"/>
         </committee>
         <committee name="Senate Committee on Small Business and Entrepreneurship" role="Member"/>
      </person>
      …
   </senate>
</congress>
```

To generate `percom.xml` from `congress.xml`, you have the following options:

(a) Write a Python program using SAX API (`xml.sax`).
(b) Write a Python program using DOM API (`xml.dom`).
(c) Write an XQuery.
(d) Write an XSLT program.

Please refer to the document "XML Tips" on the course website for instructions on how to write and run these programs and queries. You should validate your output file `percom.xml` against the provided `percom.dtd`, using the following command (more information on `xmllint` can be found in "XML Tips")

```
xmllint --dtdvalid /opt/dbcourse/assignments/hw3/percom.dtd --noout percom.xml
```

*You must implement two out of the four options.* For each option you implement, submit source code and output.

## Extra Credit Problem X1 (10 points)

Implement the other two options that you left out for Problem 3.