



The University of
Chicago
Department of
Computer Science

CMSC 16200 – Honors Introduction to Computer Science 2
Winter Quarter 2007
Lab #8 (02/26/2007)
Due: 02/28 (Wed) at 5pm

Name:

Student ID:

Lab Instructor:

Borja Sotomayor

Do not write in this area			
1	2.1	2.EC	TOTAL
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Maximum possible points: 20 + 10

One of the exercises in this lab has to be done in a group of 3. In particular, you will be asked to solve exercise 1 as a group.

Please include the names and student IDs of the students you worked with:

Name	Student ID



Exercise 1 <<10 points>>

In this exercise you will need to write a Relax NG Schema by looking at two sets of file: XML files that are valid according to that schema and files that are invalid. Our schema models the contents of a library catalog with books, movies, and music. An example of the file is the following:

```
<library name="J. Random Hacker's Library">
<books>
  <book id="EndersGame94">
    <title>Ender's Game</title>
    <author>Orson Scott Card</author>
    <publisher>Tor Science Fiction</publisher>
    <year>1994</year>
  </book>
  <book id="Hackers">
    <title>Hackers: Heroes of the Computing Revolution</title>
    <author>Steven Levy</author>
  </book>
  <book id="C">
    <title>The C Programming Language</title>
    <author>Brian Kernighan</author>
    <author>Dennis Ritchie</author>
    <publisher>Prentice Hall</publisher>
    <year>1988</year>
  </book>
</books>
<movies>
  <movie id="TBTNBC93">
    <title>The Nightmare Before Christmas</title>
    <director>Henry Selick</director>
    <year>1993</year>
  </movie>
  <movie id="EdWood" oscarwinner="yes">
    <title>Ed Wood</title>
    <director>Tim Burton</director>
    <year>1994</year>
  </movie>
  <movie id="DayBeast95" oscarwinner="no">
    <title>The Day of the Beast</title>
    <director>Alex de la Iglesia</director>
    <year>1995</year>
  </movie>
</movies>
<music>
  <album id="Wembley86">
    <title>Live at Wembley '86</title>
    <artist>Queen</artist>
    <year>1996</year>
  </album>
  <album id="Farewell95">
    <title>Farewell: Live at Universal Amphitheater</title>
    <artist>Oingo Boingo</artist>
    <year>1995</year>
  </album>
</music>
```



```
</album>
<album id="0skorri96">
  <title>25 kantu-urte</title>
  <artist>0skorri</artist>
  <year>1996</year>
</album>
</music>
</library>
```

When designing your schema, you should follow these steps:

- Write a first schema based on the first file. This will capture most of the syntax of the XML files.
- Validate the remaining “good” files against your schema. You will probably bump into small syntax errors that will require you to tweak your schema.
- Once all the “good” files validate, move on to the bad files. Make sure your schema rejects them all. If any of the bad files is accepted by your schema, take a good look at the XML file to notice how it differs from the known “good” files.



Exercise 2 <<10 + 10 points>>

We have been asked to develop a piece of software for *Uncle SNAFU's Computer Store*, a store that, being on the forefront of technology, uses XML for all its information storage and processing needs. In particular, we will be concerned with two XML documents produced by the existing POS (Point of Sale) software installed at Uncle SNAFU's.

The inventory file

This file keeps track of all the products in the store. This is a sample file:

```
<?xml version="1.0"?>
<shop name="Uncle SNAFU's Computer Store">
  <item id="1" stock="7" card="y">
    <name>FOOBAR keyboard</name>
    <price>39.99</price>
  </item>
  <item id="2" stock="15" card="y">
    <name>Grouchobyte hard drive</name>
    <price>149.99</price>
  </item>
  <item id="3" stock="81">
    <name>Wumpus repellent</name>
    <price>49.99</price>
  </item>
</shop>
```

For each product, the following information is stored:

- *Product ID*: Each product is assigned a unique id.
- *Items in stock*: The number of products immediately available in the store.
- *Name*: The product's name.
- *Price*: The product's price.
- *Discounted product*: Uncle SNAFU offers a discount program for frequent customers. Upon presenting their membership card, customers automatically get a 15% discount, but *only* if the product is marked as a discount product.

The transactions file

This file keeps track of all the transactions that have taken place during a single day. There are three types of transactions:

- *Sale transaction*: Produced when a product is sold. The transaction includes the



product ID of the item sold and specifies whether the client presented a membership card.

- *Restocking transaction*: Produced when a shipment arrives. The transaction includes the product ID of the items received and the number of items.
- *Removal transaction*: Produced when Uncle SNAFU decides to eliminate a product from the inventory, for reasons known only to Uncle SNAFU himself.

All transactions also include a timestamp.

The following is an example of a transaction file:

```
<?xml version="1.0"?>
<transactions date="02/27/2006">
  <sale itemID="2" timestamp="468822992" card="y"/>
  <sale itemID="11" timestamp="468823002"/>
  <sale itemID="7" timestamp="468823039"/>
  <sale itemID="18" timestamp="468823067"/>
  <sale itemID="13" timestamp="468823101" card="y"/>
  <restock itemID="5" timestamp="468823134" num="10"/>
  <sale itemID="7" timestamp="468823155" card="y"/>
  <sale itemID="5" timestamp="468823167"/>
  <sale itemID="10" timestamp="468823199" card="y"/>
  <sale itemID="14" timestamp="468823203"/>
  <remove itemID="8" timestamp="468823209"/>
</transactions>
```

Our job is to write a Python script that will process the transaction file at the end of the day. Our first version (<<10 points>>) will only process the sale transactions. We are interested in computing the total sales (in dollars) for the entire day, and printing out warning messages for the following conditions:

- If the inventory file states that there are no items in stock for that product, this is not an error. Quite possibly, someone made a mistake when counting the products. However, this *does* merit checking the inventory, so the program must output the following message:

**Warning: Inventory shows no stock for product '*product_name*'.
Recheck inventory.**

- If, after the sale, there are not items in stock for that product, the program must output a warning message so the manager will remember to restock that product:

Warning: '*product_name*' stock depleted

- If the client used his membership card, but the product is not a discount product,



the program must show a warning:

Attempt to use discount card on non-discount product (*product_name*)

After all the sale transactions have been processed, the program will print out the total sales (in dollars) for that day. When computing this amount, we must take into account if each sale took place at the list price or at a discounted price.

Note that, in this version of the exercise, you will have to load the inventory file, but do not need to modify it (e.g., you do not need to decrease the number of items in stock after a sale).

You can assume that both the inventory and transaction file are *coherent*. In other words, you can assume that a product ID in the transaction file always appears in the inventory file, that all values are valid (floating point numbers, integers, etc.), etc.

For extra credit (<<10 points>>), you will have to modify the inventory file after each transaction:

- If the transaction is a sale transaction, the number of items in stock must be decreased by one (except when the number of items in stock is zero).
- If the transaction is a restocking transaction, the number of items in stock must be increased by the number shown in the transaction.
- If the transaction is a removal transaction, the product must be removed from the inventory.

At the end of the program, the modified inventory must be saved to a file (the name of this file is specified with a command-line parameter; see below)



Notes

- You are provided with the following:
 - A sample inventory file, and its Relax NG Schema.
 - A sample transactions file, and its Relax NG Schema.
 - The expected inventory file resulting from correctly processing the sample files.
- Your script must be called `snafu.py`. This script will accept three command line parameters: the inventory file, the transactions file, and the name of the new inventory file (for the extra credit portion of the lab).
- We recommend you use ElementTree to solve this exercise. To do so, you can use Python 2.5, available on the CS machines in `/opt/python/python-2.5/bin/python`

Expected output

Without extra credit:

```
Warning: Attempt to use discount card on non-discount product (FOOBAR USB drive)
Warning: 'FOOBAR drive' stock depleted
Warning: Attempt to use discount card on non-discount product (FOOBAR USB drive)
Warning: Inventory shows no stock for product 'Blinkenlights'. Recheck inventory
Warning: Attempt to use discount card on non-discount product (FOOBAR joystick)
Warning: 'Grok mouse' stock depleted
Warning: 'FOOBAR drive' stock depleted
Total sales for today: 1778.09
```

With extra credit:

```
Warning: Attempt to use discount card on non-discount product (FOOBAR USB drive)
Warning: 'FOOBAR drive' stock depleted
Warning: Attempt to use discount card on non-discount product (FOOBAR USB drive)
Warning: Inventory shows no stock for product 'Blinkenlights'. Recheck inventory
Warning: Attempt to use discount card on non-discount product (FOOBAR joystick)
Warning: 'Grok mouse' stock depleted
Total sales for today: 1778.09
```