CSCI 3020 ASSIGNMENT 4 (100 POINTS)

DUE TUESDAY, OCTOBER 20, 11:59 PM

GOALS

This assignment has you create a DTD along with test XML files that demonstrate the DTD is correct.

There is no C++ in this assignment. It is 100% XML and DTD.

TIPS AND GETTING HELP

- Read these instructions closely. There is a lot of detail and you must follow it all.[1]
- Post general questions on the discussion board in D2L. Please refrain from posting more than 1 line of code, otherwise your post may be removed.
- Send me email, nicholsonja@apsu.edu, or come to my office. I'll be happy to help. If emailing me, please do not send me a code snippet; send your entire .dtd and .xml so that I can compile and test it.

BACKGROUND

You recently joined a new startup called Read Everything Today, which has decided to compete against feedly, https://feedly.com. Like feedly, your company wants to provide news feeds to users for reading content streams such as blogs, newspapers, and other sites that publish content.

Your manager has told you that part of your responsibilities is to create the DTD for the XML that will be used to push feeds to the Read Everything Today app. Your manager also wants examples of good XML that validate against the DTD and samples of bad XML that fail validation against the DTD. These good and bad samples will be used to train the other developers in the company on the required XML format.

[1] Despite what you may think, this document is small. Go look at the definition of XML, XHTML, SVG, DocBook, etc. for examples of large, complicated documents. Because of the complexity of documents like those and many others, one skill all technical people must develop is attention to detail when reading requirements and when implementing those requirements. Hopefully, assignments like this one help you see the importance and necessity of those developing those skills.

REQUIREMENTS

Create a directory for this assignment, and place your DTD file and all the 5 required XML files in that directory. You should create the following six files:

- **ret.dtd**
  - This is the DTD file for this language
  - See **The DTD Requirements** for the required contents of this file
- **valid_1.xml** and **valid_2.xml**
  - These are two examples of well-formed XML that pass validation with your DTD.
  - See **The XML Requirements** for the required contents of this file
- **bad_1.xml** , **bad_2.xml** , and **bad_3.xml**
  - These are three examples of well-formed XML that DO NOT pass validation with your DTD.
  - See **The XML Requirements** for the required contents of this file

The DTD Requirements

The language we are defining already has a set of requirements given to you by the Grace Hopper, the Extremely Senior Software Engineer at the company. Use her requirements below to write the declarations in ret.dtd.

IMPORTANT: The element, attribute, entity names, and values that are bolded must be exactly the same in your DTD – dashes and letter case matter. Otherwise, when we grade your code, it will not correctly parse the files we will be testing with.

1. The element **ret**
   - 1.1. The element **ret** has an attribute **version** that must have the value **1.0**
   - 1.2. The element **ret** has a required attribute **date** that has a text representing the date of feed.
   - 1.3. The element **ret** can have 0 or more **feed** elements
   - 1.4. The element **ret**'s last element is a required **doc-copyright** element. The copyright statement for your company.

2. The element **feed**
   - 2.1. The element **feed** starts with a required **source** element. Must appear exactly once. The source of the feed
   - 2.2. The element **feed**'s second element **feed-desc** is optional, but if present, appears no more than once. This is a description of the feed and its source.
   - 2.3. The element **feed**'s third element **info** is optional, but if present, appears no more than once. This is additional information about the feed.
   - 2.4. The element **feed**'s last element is a required **stories** element. Must appear exactly once. A list of feed stories.

3. The element **source**
   - 3.1. The element **source** has a required **name** element. Must appear exactly once.
   - 3.2. The element **source** has a required **url** element. Must appear exactly once.
     - 3.2.1. These two elements can appear in either order, **name** first or **url** first

4. The element **feed-desc**
   - 4.1. The element **feed-desc** contains at least 1 **par** element. May contain more than one. Represents paragraph(s) of text.

5. The element **stories**
   - 5.1. The element **stories** contains 0 or more **story** elements.

6. The element **story**
   - 6.1. The element **story** has an enumerated attribute **updated** that can only have the values **true**, **false**, or **unknown**. The default is **true**. Represents whether or not the story was updated since the last time the feed was downloaded.
   - 6.2. The element **story**'s first element is the **title** element. Must appear exactly once.
   - 6.3. The element **story**'s second element is the **url** element. Must appear exactly once. A link to the story.
   - 6.4. The element **story**'s third element is the **preview** element. Must appear exactly once. A short introduction to the story. Usually, about 1 sentence.
   - 6.5. The element **story**'s element **content** is optional, but if present, appears no more than once. The story. Some sites allow the story text in feeds, some don't. That is why it is optional
   - 6.6. The element **story**'s element **video** is optional, but if present, appears no more than once. A link to an optional video.
     - 6.6.1. The order of the elements **content** and **video** does not matter. If both are present, **content** can be first or **video** can be first.
   - 6.7. The element **story**'s last element **publication-date** is optional, but if present, appears no more than once and must be the last element. The date the story was published.

7. The element **title**
   - 7.1. The element **title** can contain text

8. The element **preview**
   - 8.1. The element **preview** can contain text, **break** elements, **center** elements, **bold** elements, **italic** elements.

9. The element **content**
   - 9.1. The element **content** contains at 0 or more **par** elements. This is/are a paragraph(s) of text.

10. The element **video**
    - 10.1. The element **video** can contain text

11. The element **publication-date**
    - 11.1. The element **publication-date** can contain text

12. The element **par**
    - 12.1. The element **par** can contain text, **break** elements, **center** elements, **bold** elements, **italic** elements.

13. The element **info**
    13.1. The element **info** has an optional **updated** element. If present, appears exactly once. Represents the last day and time the feed was updated on your company's servers.
    13.2. The element **info** has an optional **copyright** element. If present, appears exactly once. Represents the copyright holder of the feed data.
    13.3. The element **info** has an optional **location** element. If present, appears exactly once. Represents the physical location of the company with the feed.
        13.3.1. These three elements can appear in any order, and all three are optional.

14. The element **updated**
    14.1. The element **updated** can only contain text.

15. The element **copyright**
    15.1. The element **copyright** can only contain text.

16. The element **location**
    16.1. The element **location** can only contain text.

17. The element **doc-copyright**
    17.1. The element **doc-copyright** has an attribute **legal-statement** that must have the value **This feed and its format is copyright RET**
    17.2. The element **doc-copyright** is an empty element

18. The element **break**
    18.1. The element **break** is an empty element. Represents a forced line break in text.

19. The element **center**
    19.1. The element **center** can contain text, **break** elements, **bold** elements, **italic** elements. Represents center formatting of text.

20. The element **bold**
    20.1. The element **bold** can contain text, **break** elements, **center** elements, **italic** elements. Represents bold text formatting

21. The element **italic**
    21.1. The element **italic** can contain text, **break** elements, **center** elements, **bold** elements. Represents italic text formatting

22. The element **name**
    22.1. The element **name** can only contain text.

23. The element **url**
    23.1. The element **url** can only contain text.

24. Define the following character entities
    24.1. **author** with a definition which should be your name
    24.2. **company** with the definition **Read Everything Today**
    24.3. **homeurl** with the definition **http://www.readeverythingtoday.com**
    24.4. **nbsp** with the definition **&#160;**
    24.5. **cents** with the definition that allows the cent sign: ¢
    24.6. **copyright** with the definition that allows the copyright sign: ©
        24.6.1. For **cents** and **copyright** you must not copy/paste the symbols above into your DTD. Encode them properly so that ASCII systems could use them.

All elements and attributes declarations should be documented in the DTD. To document them take the 2 and 3 digit rules above and copy/paste them into the DTD in a comment immediately of the element or attribute declaration associated with them. For example, the comment immediately above the element definition for the **info** element would look like this:

```
<!--
     13.1. The element info has an optional updated element. If
           present, appears exactly once. Represents the last day and
           time the feed was updated on your company’s servers.
     13.2. The element info has an optional copyright element. If
           present, appears exactly once. Represents the copyright
           holder of the feed data.
     13.3. The element info has an optional location element. If
           present, appears exactly once. Represents the physical
           location of the company with the feed.
                13.3.1. These three elements can appear in any order,
                        and all three are optional.
-->
<!ELEMENT info ...
```

The purpose of these comments is to help you make sure you do everything that is required for each element and attribute. Consider them required in this assignment.

The XML Requirements

You manager wants sample files to train the other developers in the company on your XML feed language. Use the guidelines below.

1. The DOCTYPE for all XML files should be for a system DTD that references ret.dtd. The root element is always **ret**

2. Create two well-formed XML that pass validation with your DTD. Name one **valid_1.xml** and the other **valid_2.xml**
   - 2.1. Both must be well formed.
   - 2.2. Contents must be different, in order to show that you are testing different rules in the DTD.
     - 2.2.1. **valid_1.xml** should have 1 feed, with at least 3 stories.
     - 2.2.2. **valid_2.xml** should have at least 2 feeds. One with 1 story, and one with 2 stories. You can add more stories or feeds for testing as long as they validate
       - 2.2.2.1. Your XML should not be a copy of the **sample.xml** I've provided to help you.
   - 2.3. You should use these files to test as many rules and rule variations as possible. Incomplete testing may mean that your DTD will not correctly validate the XML we will be grading with.

3. Create three well-formed XML that DO NOT pass validation with your DTD. Name them **bad_1.xml** , **bad_2.xml**, and **bad_3.xml**
   - 3.1. Must be well formed
   - 3.2. For each bad XML file, pick two rules from the DTD requirements, and write XML that does not pass those rules. Each bad XML file should break 2 different rules, and the rules should be different for each file. That means you will have 3 XML and break 6 rules total.

   Each XML has a required comment (see **Grading Notes)**. In the bad XML files, you must identify the two rule numbers the XML breaks. For example, suppose I chose to show how to break rules 1.1 and 17.2 in bad_1.xml. Then my comment at the beginning of the file would be:

```
<!--
     John Nicholson
     CSCI 3020 Section W1
     Fall 2015
     Assignment 4

     bad_1.xml rules broken
          1.1   The element ret has an attribute version that
                must have the value 1.0
          17.2 The element doc-copyright is an empty element
-->
```

   When this document is validated, it must generate validation errors corresponding to those two rules.

GRADING NOTES

- If you have any question about a requirement, do not assume anything. You should email me at nicholsonja@apsu.edu with your question. If you choose to assume something, and it does not match what was intended, then you may lose points.

  While I have tried not to have errors in this document, there may be an error. If I am informed of errors, I will post corrections. A professional programmer does the same thing. Professional programmers are given requirements for projects, much like you are given requirements for homework assignments. A good professional will always ask for clarification, because incorrect assumptions can cost time and money that isn't there. You should do the same.

- You can use whatever editor and validation tools you wish, however, your submissions will be graded only with jEdit. You may want to double-check your files in jEdit just to be safe.

- Follow the guidelines in the **Submitting** section. Failure to follow those guidelines properly may result in loss of points.

- If the DTD contains syntax errors that prevent jEdit from using it for validation, you will automatically lose 50 points.

  This does not mean you will lose 50 points if you were to leave out a required tag. For example, if everything else in the DTD is correct, but you inadvertently leave out a required element definition such as the **doc-copyright** element, then that would be a small error worth a small number of points, depending on the element. The reason is that would be a bug of sorts. It does not prevent the use of the DTD; rather it incorrectly misses one element. All other elements could still be validated.

  An example of a syntax error that would cause you to lose the 50 points would be something like this:

  ```
  <!ELEMNT p (  #PCDATA ) >
  ```

  Here, ELEMENT is spelled incorrectly. This would prevent any of the DTD from being used for any validation. Think of this like writing a C++ file that cannot be compiled – kind of useless.

- If any XML file is not well-formed, you will lose 20 points. You should check that each XML document you upload is well-formed. Make sure it can pass jEdit's Parse as XML.

- You must include a comment at the top of your DTD file and every XML file in this format (DTD comments are the same as XML):

```
<!--
     John Nicholson
     CSCI 3020 Section W1
     Fall 2015
     Assignment 4
-->
```

  If this comment is missing from any of your files, you will receive 0 points on the assignment. You should obviously change the name to your name, the section to your section, the system to the system you used. You must include the specific operating system and the editor you used.

  Please note that Windows is not an operating system; Windows 7, Windows 8, and Windows 8.1 are operating systems.

TOOLS

You can use a text editor for this assignment, but you will have much more success with a programming editor that supports XML and DTD validation. The two recommendations I have given are

- Notepad++ (Windows only)
- jEdit (any operating system, Java needs to be installed).

Both of the editors have plugins available for XML. There is a video, 26_setting_up_editors, in D2L that will walk you through installing the plugins in both editors. If you wish to use another editor, feel free. Just make sure to mention it in your required comments.

Tips

- Notepad++ is more user friendly that jEdit, but it appears Notepad++'s validator is not as good as the one in jEdit. The main problem I have found is that the Notepad++ editor does not properly handle character entities. Therefore, if you choose to use Notepad++, you may want to double-check your files in jEdit for correctness before submitting.

- In jEdit, if you change the DTD, you must close any open XML file and then reopen in order to use the updated DTD. Otherwise, jEdit uses the old version that will give you incorrect results.

SAMPLE

I am providing a sample well-formed XML file, sample.xml. It is missing the DOCTYPE declaration, and does not show every aspect of the language. Currently, it is not well-formed, since it uses character entities that are not part of the XML standard. Once you define those entities in your DTD, it should pass.

It should help you understand how the requirements above should match the required language definition.

SUBMITTING

Zip up your .DTD file and all your .XML files into one .ZIP file. Upload your .ZIP file to the Assignment 4 Dropbox in D2L.

Do not upload your files individually; it will prevent your code from working when we test it due to the way D2L works when we download your files for grading. You must upload all your files in one .ZIP file. If you want to resubmit, re-zip all your files and upload a single .ZIP.   Do not submit a .RAR file. It will not be accepted. Uploading a .RAR will be considered equivalent to no submission.