COSC 3311 Software Design
Report 2: XML Translation
Due: Thursday, November 4, 7:00pm
Work in groups of size 1 or 2

1 Given

The report2 Application sub-directory contains the following files.

- `all_translations.e` and `translator.e`, which should not be changed.
- The file `FlexOrPs.header` – a Postscript file that contains the Postscript definitions that you need to make your output a proper Postscript file.
- An example input file `example-input.xml` showing some of the components in the XML files translated by the system you are to develop.
- An example output file `example-output.ps` shows a prototype of the Postscript translation of `example-input.xml`. Look at lines 548 to 624, the earlier lines are a copy of `FlexOrPs.header`. The output was manually translated and, as a consequence, is not an exact translation as specified in the specification – display line lengths were not calculated, they were estimated instead – but the output is close enough to understand what is required. Comments, also added manually, describe the translation. The output file can be displayed with ghostview on Prism.

2 Input and output structures

2.1 Input file structure

The input file is a tagged ASCII file. The tags are a small subset of the XML/HTML tags used in web documents.

The input files have the following EBNF description.

Legend:	[A]	– optional A, can have or not have
	[A , B]	– optional choice: can have A or B or neither
	+[...]	– zero or more iterations of the choice – can have a different choice for each iteration
	(A)	– must have an A
	(A , B)	– must have one of A or B
	+(...)	– one or more iterations of the choices – can have a different choice for each iteration
	A B	– have an A followed by a B; there is a space between the A and B

The input file, viewed as a sequence of ASCII characters, has the following structure.

```
File        ::= +[ Line ] Eof ;
Line        ::= [ [ Space ] Tag [ Text ] , Text ] Eol ;
Tag         ::= '<' TagName '>' ;
Text        ::= +( Word , Space) ;
Word        ::= +( GraphicCharacter ) ;
Space       ::= +( ' ' ) ;
GraphicCharacter  ::= Any ASCII character that puts ink on paper -- no whitespace or control
                      characters.
TagName  ::= ( "SECTION" , "/SECTION" , "P", "UL", "LI", "/UL") ;
```

The input file, viewed as a sequence of tagged components, has the following structure, which uses some of the previous rules.

```
File        ::= +[ Section ] Eof ;
Section     ::= "<SECTION>" SectionHeader "</SECTION>"
                +[ Text , TaggedComponent ] ;
SectionHeader  ::= +[ Text ]
TaggedComponent   ::= +( Paragraph , BulletedList ) ;
Paragraph     ::= "<P>" [ Text ] ;
BulletedList  ::= "<UL>" +[ Text , TaggedComponent , ListItem ] "</UL>" ;
ListItem      ::= "<LI>" +[ Text , TaggedComponent ] ;
```

2.2 Output file structures

Many XML files have multiple translations depending upon the information to be extracted and displayed. Your system is to produce two translations of an input XML file.

2.2.1 ASCII output

The ASCII output is to be an exact copy of the input – i.e. executing the diff command

```
diff input.xml output.ascii
```

will show no difference between the input and the output.

Creating the identity translation is important for the following reasons.

- It tests the design of your system such as your input classes, internal representation classes, and output classes with simple processing algorithms on the output side.
- It is simple to test if your program produces correct output using the diff command.
- It provides a default translator that can be adapted for other needs

2.2.1 Postscript output

The output is a Postscript file that can be: (1) viewed using ghostview; (2) printed; or, (3) distilled, using Acrobat distiller, to produce a pdf file.

The Postscript output file is constructed in the following way by processing the tags in the order in which they appear.

- The file `FlexOrPs.header` is copied to the output file
- An initialization sequence is output. You change the header date and head title that appear within the parenthesis to customize the date and header .

  ```
  "%%%%%%%%%%%%%%%%%%%%%%  Separate header from data
  /PageNumber 1 def
  /HeaderDate (1904 January) def
  /HeadTitle (A dummy title for the header line) def
  headerLine topOfPage TextFont"
  ```

- The tag <SECTION> outputs the string

  ```
  " HeadFont2 textMode"
  ```

  Command strings are output without the quotes.
- The tag </SECTION> outputs the string

  ```
  " nl HeadFont2size 5 sub vtab TextFont"
  ```

  Each command string has a space character at the beginning so Postscript commands are separated by at least one space character from preceding commands. An end of line has the same effect.
- The tag <P> outputs the string

  ```
  " nl nl".
  ```
- The tag <UL> outputs the string

  ```
  " 20 indentLM"
  ```
- The tag </UL> outputs the string

  ```
  " -20 indentLM"
  ```

- The tag <LI> outputs the string
  " -10 indentLM nl MathFont (\267) show TextFont
  LeftMargin 10 add vpos moveto 10 indentLM "
  Can be on multiple lines or all on one line.
- EOL outputs nothing – the translator fills output lines to the maximum width. When the translator determines that the output should go to a new line the string " nl" is output.
- EOF outputs the string followed by a physical new line character
  " nl footerLine showpage"
- WORD outputs the string
  " (Graphic Characters) show".
  Backslash, "\" is the escape character.
  If a graphic character is a left parenthesis, then output the pair of characters "\(".
  If a graphic characters is a right parenthesis, then output the pair of characters "\)".
  If a graphic character is a backslash then output the pair of characters "\\".
- SPACES outputs the string
  " ( ) show"
  Only one space is output; i.e. multiple spaces in sequence are reduced to 1.

The starting separation between the left and right margins is 6.75*72 (= 486) points, which can hold approximately 100 characters of text and spaces in normal documents. This is approximately 4.86 points per character. Before indenting for list items, you need to put a new line command every 100 characters, otherwise the line will go off the right hand edge of the paper. When you output Word and Space you keep a running count of the display line length and, if the next output would be more than 100 characters, you output the string " nl" to start a new display line before you output the next word.

The % character in a Postscript file is the comment character. The remainder of the line is a comment, similar to the -- in Eiffel. You can use comments in your output to help you understand what you have output, see the example output file. To make the output files readable when you are debugging your system and for other people to verify your output, we suggest you put the Postscript commands for the tags on physical new lines (%N in Eiffel) in the output file and output Postscript comments to identify the command.

3 Implementation

Not all translations are to Postscript. There are many other possible translations where other strings or routines correspond to the tags and tag structure, which implies a structure to the document, which is not just linear. As a consequence, you need to create an internal representation that faithfully captures the inherent structure of the input file. You need to develop classes that will facilitate the creation of many different translations from the same internal representation. You need to develop a design that facilitates adding more tags; in practice, XML/HTML documents have many dozens of tags – use view source in a browser to see how many tags there are in real documents, and how complex tags can be.

Classes ALL_TRANSLATIONS and TRANSLATOR cannot be changed. You are free to modify classes XML2ASCII and XML2PS as you see fit. You can add as many other classes as necessary for your design and, where appropriate, use inheritance from your own classes or from Eiffel classes.

It is suggested you start your classes with stubs – do nothing routines and functions that return a constant value – and contracts; it is important that your classes have complete contracts. Then implement a few stubs at a time while getting more and more of the system to work. The first stubs to work at are the input and then the output using the identity translator. When that works, implement the Postscript translator.

For each class you develop you will need to have a corresponding test class in Espec style in the testing cluster. Each test case must include a call to **comment** as in Report 1. Espec type tests are particularly useful in the early going to verify that features that read the input file correctly build the internal structure. You want to be confident of what works so when a problem occurs it is easier to find the cause. Espec type testing is less useful on the output side because there it can be more effective to view, with ghostview, the output from test files.

Errors in and ill-formed user input are time consuming to handle but do not change the essential design; the corresponding grammar is more complex. As a consequence, assume the input is correct.

Assume that all tags are on one line, thus an < that represents the start of a tag has a matching > before the end of the line. Tags may have spaces before the <, for visual feedback in the XML file.

You are required to store the entire input in memory, so you will have to develop classes to represent the input. The control characters are the following, which you can use in your internal representation.

- Eol – end of line. CTRL-10 – ASCII end of line, value 10 – %N in Eiffel
- Eof – end of file. CTRL-26 – ASCII end of file, value 26
  Eiffel does not have a character for this but you can use %/26/

Remember to use referential transparency – functions have no side effects.

4 Design Document

You have a 15-page limit at a reasonable font size. Longer documents will receive a zero grade.

The design document should address such issues as the design problems you encountered, what your solution is, what your rationale is, what other alternatives there are. The design can be discussed at various levels such as an overall view, correspondence of your classes to the problems, and the purpose of the various classes that you developed.

You should use diagrams where they help your exposition. You must include at least one BON diagram of your system but you may also have sub-BON-diagrams where you can guide the user to particular issues on sub-parts of the solution.

It is very strongly recommended that you start your design document when you start working on the system and continuously update the document with your problems, alternative solutions, choices and rationale. This helps you in three ways. The first way, the document is something you can refer to to keep you on track or, if changes are required, reduce thrashing among alternative choices. The second way, is working on the document is a change of viewpoint that can help you arrive at good solutions for your design problems – a change is as good as a rest in some situations. The third way, is by the deadline you have a nearly complete or complete design document.