CIS*4650 (Winter 2020) Marking Scheme for Warmup Assignment		Marks	Comments
Token Patterns (50)	Open Tags (9)		
	Close Tags (7)		
	Words (6)		
	Numbers (8)		
	Apostrophized (7)		
	Hyphenated (6)		
	Punctuations (3)		
	Delimiters (4)		
Token Generation (35)	Correct token sequence (15)		
	Filtering Irrelevant tags & embedded text (10)		
	Reporting errors for mismatched tags (10)		
Style and Documentation (15)	Readme (4)		
	Makefile (3)		
	Testing Plan (4)		
	Coding Style (4)		
	Total (100)		

Detailed checklist:

(1) Open tags:
- One general regular expression definition rather multiple special-case definitions
- Tagnames should not be case-sensitive
- Possible to allow attribute-value pairs within a tag
- Tagnames may contain digits such as “<H1>”
- Possible to allow spaces around a tagname such as “< Doc >”

(2) Close tags:
- One general definition rather than multiple special-case definitions
- Tagnames should not be case-sensitive
- Tagnames may contain digits such as “</H1>”
- Possible to have spaces around a tagname such as “</ Doc >”

(3) Words:
- May contain both upper- and lower-case letters
- May contain digits such as “mp3” and “123abc”
- Should not contain any other punctuation marks

(4) Numbers:
- All integers
- All real numbers
- Optional “+” and “-” signs

(5) Apostrophized:
- Any sequences of strings with letters and digits, separated by apostrophes such as “John’s”, “O’Reily”, “O’Reily’s”, “world’cup”, and “this’is’just’a’test”
- Can be combined with a hyphenated such as “father-in-law’s”
- Sequences of apostrophes such as ‘’’ should be split into punctuations

(6) Hyphenated:

- Any sequences of strings with letters and digits, separated by hyphens such as “data-base”, “father-in-law”, and “this-is-just-a-test”
- Sequences of hyphens such as “---” should be split into punctuations.

(7) Punctuations:
- A filler pattern that should capture all printable punctuation marks

(8) Delimiters:
- All sequences of delimiting characters such as space, tab, and newline.

(9) Correct token sequences
- Generate output tokens reliably
- Tokens are shown with the correct and consistent format
- Relevant tags are made more specific such as OPEN-DOC, CLOSE-TEXT, and so on

(10) Filtering:
- Only the relevant tags and their embedded text are sent to the output
- You can assume that relevant sections may contain irrelevant sections, but not the other way around. This implies that you should output tokens if all tagnames on the stack are relevant; otherwise, filter the tokens if any tagname on the stack is irrelevant.
- This mechanism should be implemented inside your *.flex file so that all the details are hidden inside this module.

(11) Reporting errors:
- Display an error message for every unmatched close tag
- Display all unmatched tagnames on the stack at the end of the input
- Error messages should be sent to “stderr”, not “stdout” (which should only contain relevant information)
- Again, the global stack and related operations should be inside your *.flex file so that we can hide the implementation details.

(12) Readme:
- What does your program do
- Any assumptions and limitations
- User guide for building and testing your program
- Any possible improvements for future

(13) Makefile:
- Support “make” and “make clean” for building and re-building your program

(14) Testing plan:
- Identify all major conditions for testing your program

(15) Coding style:
- Provide adequate comments for a file, a function, and major steps
- Use meaningful names and symbolic constants
- Use consistent indentation for the control structures
- Follow modular designs (avoid doing too much in a single function)