CS 3353 Data Structure and Algorithm Analysis I
Assignment 03: Full Marks 100
Due Date: 10/23/2022, 11:59 PM CT **End Date: 10/30/2022, 11:59 PM CT**

Read the following instructions before you proceed with the assignment:

- The assignment needs to be completed individually in Java.
- Make sure the program runs in CSX machine.
- Please go through the grading rubrics and submission instructions carefully to avoid losing points.

Stack and Queue are important data structures and are extensively used in real word applications. The objective of this programming assignment is to make students understand how to combine and reuse data structures to achieve the design goal.

In this assignment, you will implement stack using queue data structure. You need to use your own list ADT and not use list from Collection frameworks i.e., you cannot use the Queue Interface from the Collections Framework.

Input File Description:

In this assignment, you can use the provided 3 test input files (*text1.txt*, *text2.txt*, *text3.txt*,) to check if your code is working correctly or not. These three test files are simple HTML page, given as .txt file extension.

For interested students: The HTML pages have .html as extension. If you want to see how these pages look like, then you can change the extension from .txt to .html and use any browser to open these files.

The building components of a web page are HTML elements. HTML elements are a collection of tags (i.e. start tag and the end tag) and attributes that are present in every HTML page.

i) A tag instructs the web browser where an element starts and stops. The HTML tag (whether it is a start tag or an end tag) begins with < symbol and end with > symbol.

   For example, `<html>`, `<head>`, `<body>`, `<h1>`, `<p>` are all start tags, while `</html>`, `</head>`, `</body>`, `</h1>`, `</p>` are its corresponding end tags respectively.

   Note: Some HTML elements (like the `<br>` and `<hr>`) have no content and they are called empty elements. `<br>` refers to the line break element while `<hr>` represents a thematic break between paragraph-level elements, for example, a change in story scene. Empty elements do not have an end tag.

   For interested students: The complete list of all the HTML elements, which are created using tags can be found on this page: https://developer.mozilla.org/en-US/docs/Web/HTML/Element. The list of empty tags can be found on the link: https://developer.mozilla.org/en-US/docs/Glossary/Void_element

ii) An HTML element is defined by a start tag, some content, and an end tag.

   For example, in

```
<TITLE>Department of Computer Science | Oklahoma State
                    University</TITLE>
```

where,
`<TITLE>` is a start tag
`Department of Computer Science | Oklahoma State University` is the element content
`</TITLE>` is an end tag

iii) An attribute describes an element's properties. It is always placed in the start tag of an element. It generally provides additional styling (attribute) to the element.

For example, in `<BODY BGCOLOR="FFFFFF">`, `<BODY>` is a start tag and within this tag you have attribute `BGCOLOR` to specify the background color, which in this case is `FFFFFF`.

Task:

You need to write a Java program that checks whether the given HTML file (given as .txt file) is created following the given tag rules or not:

1. Tags are always surrounded by angle brackets (less-than/greater-than sign), like in <HEAD>.

2. All tags (except `<br>` and `<hr>`) come in pairs and surround the material they affect. They work like a light switch: the first tag (i.e. start tag) turns the action on, and the second tag (i.e. end tag) turns it off. Please note that the end tag followed by the start tag is not a valid tag rule.

3. First tag on, last tag off. Tags are embedded, so when you start a tag within another tag, you must close that inner tag before closing the outer tag. For instance, the page will not display properly with the tags in this order:

```
<HEAD><TITLE>Your text</HEAD></TITLE>.
```

   The correct order is:

```
<HEAD><TITLE>Your text</TITLE></HEAD>.
```

4. Many tags have optional attributes that use values to modify the tag's behavior. For example, the <P> (paragraph) tag's ALIGN attribute, lets you change the paragraph alignment. For example,

```
<P ALIGN=CENTER>
```

   centers the next paragraph on the page. The presence or absence of the attributes should not affect the HTML tag rule.

If the HTML file meets all the above rules, then the program needs to display message as:

```
Congratulations...
The given HTML file meets all the tag rules..
```

If the HTML file does not meet the above rule(s), then the program needs to display the error message saying which tag in which line is causing the problem.

```
Oops... There is a problem..
The <p> tag at line # 4 does not meet the tag rules..
```

Students need to consider the following assumptions for the assignment:

i) All the test files only contain two empty tags: `<br>` and `<hr>`. For all other elements, you can consider every start tag will have its corresponding end tag.

ii) The test file can have comments. In HTML, a comment is text enclosed within <! ---> tags. This syntax tells the browser that they are comments and should not be rendered on the front end. For example, the line

```
<!--This is a comment, which will not be displayed in the
browser-->
```

is a comment. For comment, the start tag is `<!--` while the end tag is `-->`.

iii) The test file can contain the line:

```
<!DOCTYPE html>
```

as the first line. However, it is not always necessary to have this as the first line. This line is not actually an element or a HTML tag. It just lets the browser know how the document should be interpreted, by indicating the version or standard of HTML being used. Interested students can learn more about this line from the link: https://www.w3.org/QA/2002/04/valid-dtd-list.

Implementation Requirement:

You need to use a stack to check the tag rules. The stack is implemented using two queues.

Before proceeding to the algorithm, let us briefly understand the basics of these two data structures.

A. Stack

The items are added to a stack in the LIFO (Last In, First Out) order. This implies that the first element to be removed from the stack will be the last element to be added. The fundamental operations on a stack are:

i. *Push*: to add an item at the top.
ii. *Pop*: to remove an item from the top.

B. Queue

The items are added to a queue in FIFO (First In, First Out) order. This means that the first item added is also the first one withdrawn. The queue's fundamental operations are:

i. *Enqueue*: to place an item at the back
ii. *Dequeue*: to take an item from the front

To perform the stack operations like *Push* and *Pop*, you need to implement Queue(s) first. Use queue operations *Enqueue* and *Dequeue* in such a way that they are equivalent to *Push* and *Pop* operations. The logic to achieve *Push* and *Pop* operation for stack using Queue's *Enqueue* and *Dequeue* is given below:

Use two queues (`Q1` and `Q2`) to implement stack. Here, `Q1` will be the main source for the stack, while `Q2` is just a helper queue used to preserve the order expected by the stack.

A. Implementation of Stack's *push* operation using queue i.e., to push an element *E* in the stack:

- if `Q1` is empty, *enqueue E* to `Q1`
- if `Q1` is not empty, *enqueue* all the elements from `Q1` to `Q2`, then *enqueue E* to `Q1`, and *enqueue* all elements from `Q2` back to `Q1`

B. Implementation of Stack's *pop* operation using queue i.e., to pop an element from the stack:

- *dequeue* an element from `Q1`.

Lastly, you need to write the algorithms for the implementation of stack's *push* and *pop* operation and compute the worst case complexity of these operations.

The TA (Teaching Assistant) will run the program in CSX server as follows, please note the command-line argument:

Consider your file name is `Assignment03.java`.
The TA will compile the file as: `javac Assignment03.java`

which will then produce .class file as `Assignment03.class`.

The TA will then run the program using: `java Assignment03 <filename>`

where,
`Assignment03` is the .class file produced after the compilation
`<filename>` is the test HTML file that will be used as the input for the program

Example:

1. `java Assignment03 text1.txt`

will use text1.txt file as input to check the HTML tag rule.

2. `java Assignment03 text2.txt`

will use text2.txt file as input to check the HTML tag rule.

Grading Rubric:

Your assignment will be graded based on the following grading Rubrics:

• Design of Queue using list ADT		**[15 Points]**
• Design of Stack using Queue as described in the implementation requirement		**[25 Points]**
• Successful completion of the HTML tag rule verification		**[45 Points]**
• Worst case complexity analysis		**[15 Points]**
○ Algorithms for stack's *push* and *pop* operations ○ Stack's *Push* operation using *Enqueue* and *Dequeue* ○ Stack's *Pop* operation using *Enqueue* and *Dequeue*	[5 Points] [5 Points] [5 Points]	

Tips to avoid any point deduction:

- Failure to follow standard programming practices will lead to points deduction, even if the program is running correctly, like
  - No comments on code
  - Not writing meaningful identifiers
- If the program does not compile successfully on the CSX machine, then 20 points will be automatically deducted.

Submission Guidelines:

- All the files should be submitted through Canvas. Assignment submitted through email will not be accepted. CSX machine is only used to run the program and not for assignment submission.

- Your assignment will be checked in the CSX machine. You need to include readMe.txt file that shows:
  - how to run your program.
  - which csx server has been tested on
  - any assumption you made on the assignment.

- The worst-case analysis needs to be performed in a separate pdf file. It needs to be computer typed and not handwritten. Name this file as:

  assignment03_lastName_firstName_worstCaseComplexity.pdf

- All the java files need to have the following format:

  assignment03_lastName_firstName_<appropriate_file_name>.java

  where, use meaningful file name in the place of appropriate_file_name.

- In addition to .java file, all the codes should be submitted in .pdf format as well. Please copy the codes in the text form. Do not screenshot or include any image of the code in the pdf. This is used to check for plagiarism.

- The main driver file needs to have the following header information:
  - Author Name:

    - Email: <Use only official email address>
    - Date:
    - Program Description:

- Use comments throughout your program.
    - Each class and the method should be properly commented:
        - description of arguments and return data
        - description of method