COMP6481 – Fall 2021
Programming and Problem Solving
Assignment # 2
Due Date: Monday, November 1, 2021 by 11:59PM

Part I

Please read carefully: You must submit the answers to all the questions below. However, this part will not be marked. Nonetheless, failing to submit this part fully will result in you missing 50% of the total mark of the assignment.

Question 1

I)

Develop well-documented pseudo code that finds all consecutive similar elements of a given array (of any size *n)*. The code must display the start indices where the values start to repeat, as well as the values of these elements. For instance, given the following array *A*:
(22, 9, 61,61, 61, 21, 0, 9, 9, 9, 9, 35, 81,81, 9, 5, 5), your code should find and display something similar to the following (notice that this is just an example. Your solution must not refer to this particular example):

Value 61 is repeated 3 times starting at Index 2
Value 9 is repeated 4 times starting at Index 7
Value 81 is repeated 2 times starting at Index 12
Value 5 is repeated 2 times starting at Index 15

a. Briefly justify the motive(s) behind your design.
b. What is the Big-O complexity of your solution? Explain clearly how you obtained such complexity.
c. What is the Big-Ω complexity of your solution? Explain clearly how you obtained such complexity.
d. What is the Big-O *space* complexity of your solution?

II) Develop a well-documented pseudo that solves the problem stated in part 1), using either a stack *S* or a queue *Q* to perform what is needed.

a. Briefly justify the motive(s) behind your design.
b. What is the Big-O complexity of your solution? Explain clearly how you obtained such complexity.
c. What is the Big-Ω complexity of your solution? Explain clearly how you obtained such complexity.
d. What is the Big-O *space* complexity of the utilized stack or queue? Explain your answer.

Question 2

Assume that we have one single array A of size N, where N is an even value, and that the array is not expandable. We need to implement 2 stacks. Develop well-documented pseudo code that shows how these two stacks can be implemented using this single array. There are two cases you must consider:

Case I: Fairness in space allocation to the two stacks is required. In that sense, if Stack1 for instance use all its allocated space, while Stack2 still has some space; insertion into Stack1 cannot be made, even though there are still some empty elements in the array;

Case II: Space is critical; so you should use all available elements in the array if needed. In other words, the two stacks may not finally get the same exact amount of allocation, as one of them may consume more elements (if many push() operations are performed for instance into that stack first).

For **each** of the two cases:

e. Briefly describe your algorithm;
f. Write, in pseudocode, the implementation of the following methods, for each of the stacks: push(), pop(), size(), isEmpty() and isFull();
g. What is the Big-O complexity for the methods in your solution? Explain clearly how you obtained such complexity.
h. What is the Big-**Ω** complexity of your solution? Explain clearly how you obtained such complexity.

⇨ Is it possible to solve the same problem, especially for Case II, if three stacks were required? If so, do you think the time complexity will change from your solution above? You do not need to provide the answer to these questions; but you certainly need to think about it!

Question 3

An Antique dealer needs to store records of the items in the inventory into a stack (the dealer believes that keeping items longer will be beneficial; nonetheless, the business logic here is not important!). Besides the usual push() and pop(); the dealer needs to always know what is the most expensive item in the inventory/stack. You are hired to implement a new method, called **max()**, which returns the current maximum value for any item in the stack. Notice that the stack grows and shrinks dynamically as elements are added or removed from the stack (i.e. the contents of the stack are not fixed).

a) Write the pseudocode of the max() method.
b) What is the Big-O complexity of your solution? Explain clearly how you obtained such complexity.
c) Is it possible to have **all** three methods (push(), pop() and max()) be designed to have a complexity of **O(1)**? If no; explain why this is impossible. If yes; provide the pseudocode of the algorithm that would allow O(1) for all three methods (this time, you do not only need to think about it, but to actually give the pseudocode if you believe a solution is feasible!)

Question 4

Is it possible to have a single tree with the following traversals? If yes, draw the tree. If not; explain which elements prevents you most from having one such tree.

- Each internal node of T stores a single character;
- A preorder traversal of T yields: E K D M J G I A C F H B L;
- A postorder traversal of T yields: D J I G A M K F L B H C E.

Part II

Purpose: The purpose of this assignment is to allow you practice Exception Handling, and File I/O, as well as other previous object-oriented concepts.

Java**S**cript **O**bject **N**otation or **JSON** is a well-known, lightweight data-interchange format, which is easily readable/writable by humans [1][2]. It is also easy for machines to parse and generate. JSON is based on data objects consisting of attribute–value pairs and array data types. It is widely used for asynchronous browser–server communication, including as a replacement for XML in some AJAX-style systems. As such, many journals and scientific servers are supporting this format. This format is easily extendable. The following is a typical sample (modified from an existing IEEE paper).

@ARTICLE{

8247289,
author={J. Park and J. N. James and Q. Li and Y. Xu and W. Huang},
journal={IEEE Transactions on Vehicular Technology},
title={Optimal DASH-Multicasting over LTE},
year={2018},
volume={PP},
number={99},
pages={15-27},
keywords={Forward error correction;Long Term Evolution;Maintenance engineering;Multicast communication;Resource management;Static VAr compensators;Streaming media;DASH;LTE;convex optimization;eMBMS;multicasting},
doi={10.1109/TVT.2018.2789899},
ISSN={0018-9545},
month={January},
}

In JSON, the fields however do not need to be placed in specific order. For example, in the above example, you can have the “number” filed for instance, be written above the “year” field, and so on.

There are many parsers available for JSON developed in may programming language as library, API, etc. Mendeley [3] is one example that uses this format. With Mendeley you can create your own library from all articles that you have read so far and you can use them when you want to write an article. Consequently, this is very good for inserting the needed references on a published paper and managing them afterwards. In particular, such tools can import the article(s) information from a bibliography file (.bib) and generate the reference(s) in particular format according to the conference or journal publisher standard.

For example, the representation of this file in IEEE format would be (this is a slightly simplified version than the actual one; however you need to follow this format for the scope of this assignment):

J. Park, J. N. James, Q. Li, Y. Xu, W. Huang. "Optimal DASH-Multicasting over LTE", IEEE Transactions on Vehicular Technology, vol. PP, no. 99, p. 15-27, January 2018.

Or in **ACM** format would be:

[1] J. Park et al. 2018. Optimal DASH-Multicasting over LTE. IEEE Transactions on Vehicular Technology. PP, 99 (2018), 15-27. DOI:https://doi.org/10.1109/TVT.2018.2789899.

And finally, in **Nature Journal (NJ)**, which is one of the most famous journals in natural science, would be:

J. Park & J. N. James & Q. Li & Y. Xu & W. Huang. Optimal DASH-Multicasting over LTE. IEEE Transactions on Vehicular Technology. PP, 15-27(2018).

You should have a "very" detailed look at these formats to see how they are mapped from the original record of the article.

In this assignment, you will be designing and implementing an alternative tool (to the existing ones), called ***BibCreator***. The main task of this tool is read and process a given .bib file (which has one or more articles) and create **3** different files with the correct reference formats for IEEE, ACM and NJ.

In short, you are given 10 files, called *Latex1.bib* to *Latex10.bib*. You BibCreator application will need to read all 10 input files (in one execution), determine whether each of these files is valid or not (details given below). If a file is valid, then BibCreator will create 3 different files based on the articles in this file, one for IEEE format, one for ACM format and the last for NJ format. To distinguish our application from other existing similar software, we will call these files IEEE***i***.json, ACM***i***.json, and NJ***i***.json (although in fact, the created files contain the references to the articles, and not json records!), where ***i*** is the Latex file #. For instance, Latex3.bib will result in the creation of 3 files called: IEEE3.json, ACM3.json and NJ3.json. If a file is invalid, then none of the 3 output reference files (for this invalid file) is created. So, in best case, BibCreator execution will result in the creation of 30 output files if all given Latex files are valid, and in worst case it will create 0 files (when all 10 input files are invalid).

The fine details of what you need to do and how your BibCreator should work are given below.

1. For the purpose of this assignment, and to provide little simplifications, the following should be assumed in relation to the input files and the articles (records) in these input file:
    a. Each file may have one or more articles; the number is unknown before processing, and your code must assume that;
    b. An article starts with @ARTICLE followed by the body of the article (between "{" and "}"). It is assumed that all articles have enclosing bodies;
    c. Inside each of these article, there exists few fields; i.e. author, volume, year, etc. These fields are assumed to always start with the field name, followed by an "=" sign and a "{" character. For instance: pages={ , month= , etc.
    d. It also assumed that each of the bodies of these fields have a closing character for its body. In specific, it is assumed that each of these fields end with "},";
    e. For simplicity, it is also assumed that there is a doi field, which maps to https://doi.org/
    f. The order of the article is NOT important. I.e., it is okay to have any of these fields above or below other fields;
    g. It is also assumed that empty lines can be there within the body of the articles, as well as between different articles;

h. HOWEVER (read carefully), any of the input files may have some of these fields as empty; i.e. number={}, title={}, etc. This is what we classify as an Invalid File. In other words, for a file to be valid, the file cannot have an empty field at any of its articles. Below is an image of an invalid file.

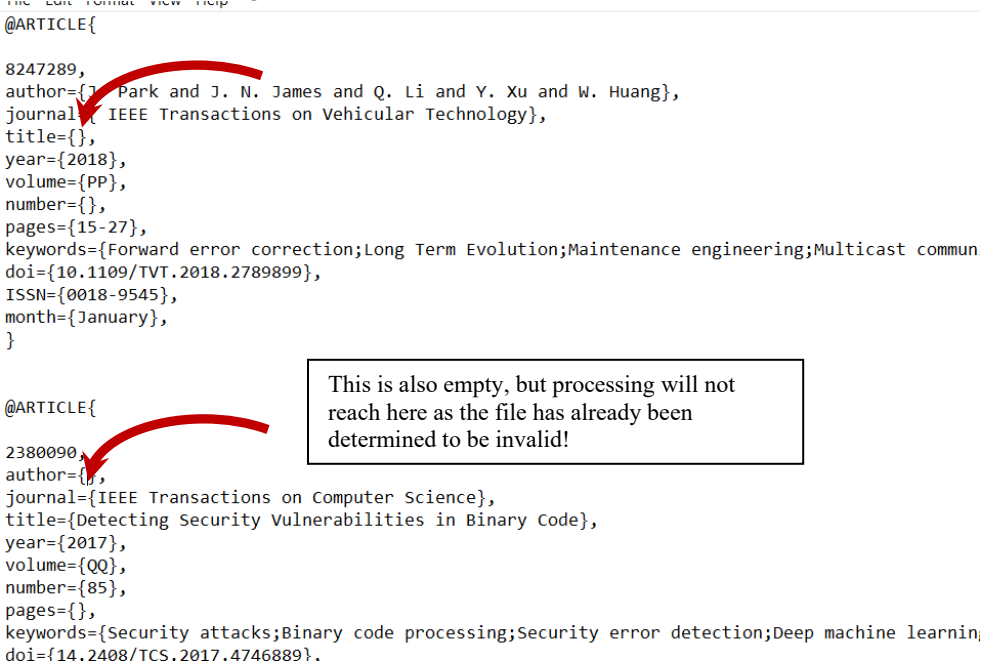

Figure 1. Example of an Invalid Input File

2. Write an exception class called **FileInvalidException** exception. The class should have sufficient constructors allow:
   a. A default error message "*Error: Input file cannot be parsed due to missing information (i.e. month={}, title={}, etc.*) " to be stored in the thrown object; and
   b. The passing of any different error message if desired. This is actually the constructor that you will be using throughout the assignment (see Figure 3 below).

3. In the main() method of the BibCreator class, attempt to open all 10 input files (Latex1.bib to Latex10.bib) for reading. You need to use the **Scanner** class for the reading these files. If "any" of these files does not exist, the program must display an error message indicating "*Could not open input file xxxxx for reading. Please check if file exists! Program will terminate after closing any opened files.*", and then exits. You MUST however, close all opened files before exiting the program. For example, if *Latex3.bib* does not exist, then the following image shows the behavior of the program.

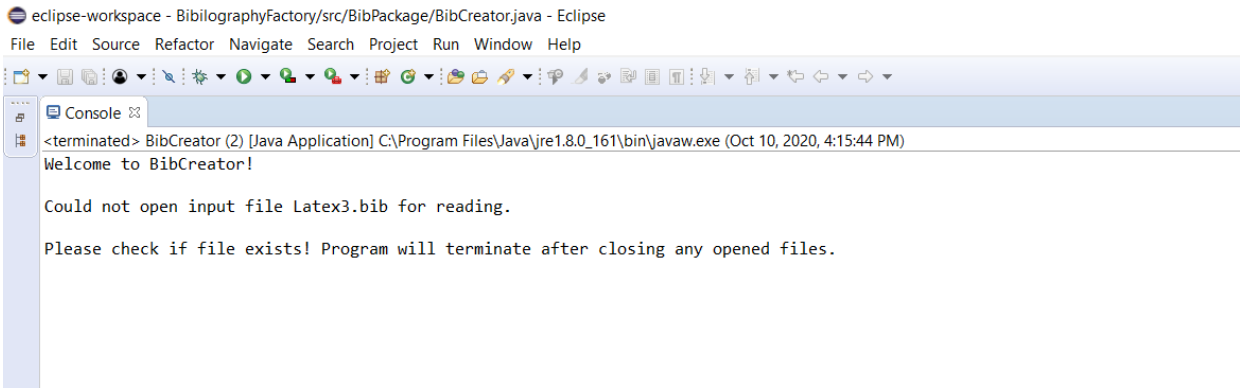

Figure 2. Example of Program Termination – One of the Input Files does not Exist

4. If all 10 input files can successfully be opened, the program will attempt to open/create all 30 output files (IEEE1.json to IEEE10.json, ACM1.json to ACM10.json, and NJ1.json to NJ10.json). You need to use PrintWriter to open these output files. If "any" of these output files cannot be created, then you must:
    a. Display a message to the user indicating which file could not be opened/created;
    b. Delete all other created output files (if any). That is, if you cannot create all of these output files, then you must clean the directory by deleting all other created files;
    c. Close all opened input files; then exist the program.

If you reach this step, then all 10 input files have been opened and all 30 output files have also been created (however, they are surely empty).

5. Write a method (you should take advantage of static throughout the entire assignment!) called **processFilesForValidation**. This method will represent the core engine for processing the input files and creating the output ones. You can pass any needed parameters to this method, and the method may return any needed information. This method however must NOT declare any exceptions. In other words, all needed handling of any exceptions that may occur within this method, must be handled by the method. In specific:
    a. The method should work on the already opened files;
    b. A method must process each of these files to find out whether it is valid or not;
    c. If a file is valid, then the method must create the proper records for each of the 3 formats (IEEE, ACM and NJ) and store them in these files;
    d. If a file is invalid, then the method must stop the processing of this file only, throws **FileInvalidException** to display the exception error message, then display a message indicating which file was detected as invalid, and where the "first" problem in that file was detected (See Figure 3). The corresponding output file MUST then be deleted;
    e. The method will then continue with the processing of the following file.

For instance, let us assume that the given input files (these files can be any files and they are not restricted to the ones provided with the assignment; in fact, the marker will execute your assignment with different files; so your code must work correctly for any given files) have 3 invalid files, Latex3.bib, Latex4.bib and Latex8.bib. Your program must detect these invalid files and show the following:

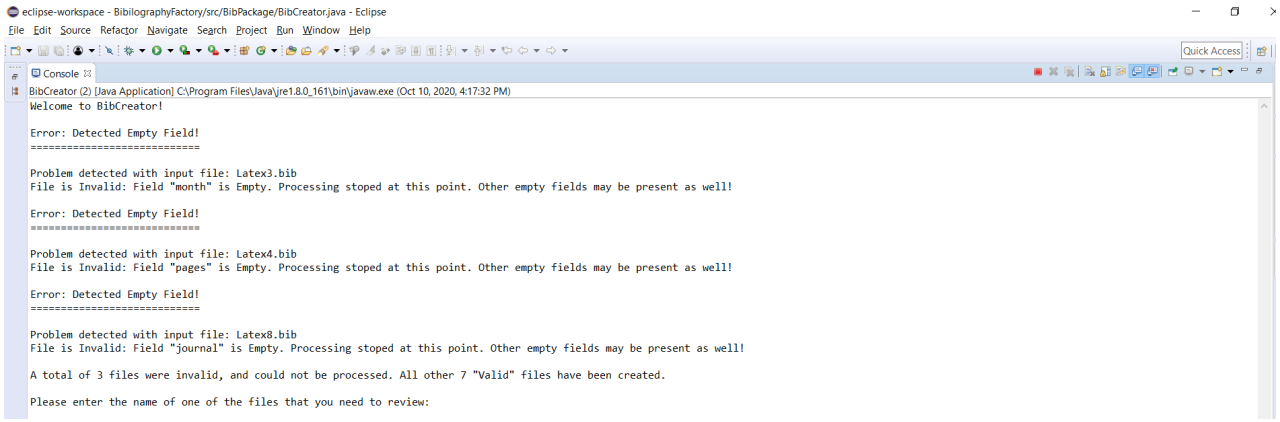

Figure 3. Example of Processing – Number of Invalid Files and Particular Fields are Indicated

6. Again, once the processing is done, all unsuccessfully created files MUST be deleted. Here is how the directory would look like at this point based on the above senario:

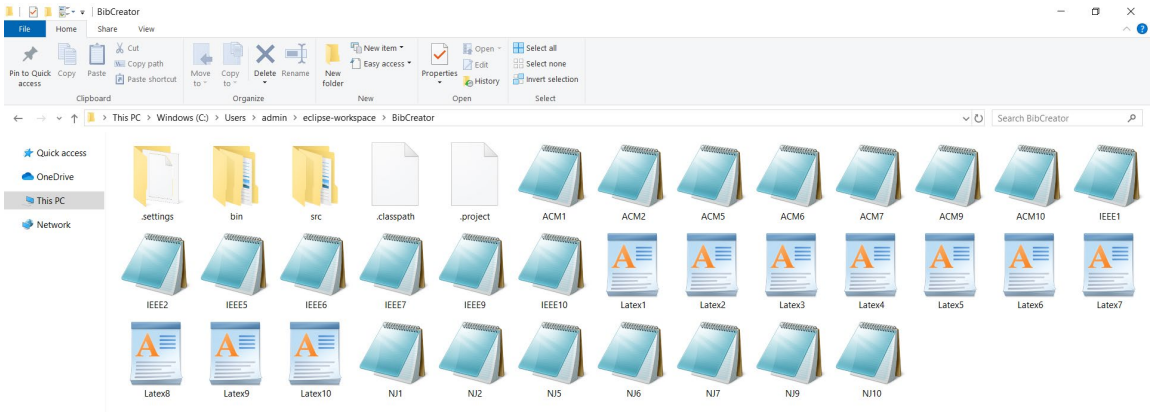

Figure 4. Contents of Current Directory after above processing Example

7. Finally, at this point, the program needs to ask the user to enter the name of one of the created output files to display. If the user enters an invalid name, a FileNotFoundException should be thrown; however, the user is allowed a second and final chance to enter another name. If this second attempt also fails, then the program exists. Figure 5 and Figure 6 below show the behavior of this program. You must however apply the following:
    a. If the entered file is valid, then your program must open this file for reading using the **BufferedReader** class. Do not use the Scanner class to read the file for that task.

8. Finally, here are some general information:
    a. It may assist you greatly if you take advantage of static variables/attributes and static attributes throughout the assignment; in fact, it is not necessary to utilize other aspects such as Inheritance, Polymorphism, etc.
    b. You must exactly match the format and look of the expected output files. For instance, use & or et al. for the authors as expected, follow the exact order/format of the contents,

use vol. instead of volume when as expected. In other words, a small difference in the expected output will surely result in mark deduction;

c. For the processing of the authors, you may want to use the **StringTokenizer** class;
d. You should minimize opening and closing the files as much as possible; a better mark will be given for that;
e. Do not use any external libraries or existing software to produce what is needed; that will directly result in a 0 mark!
f. Again, your program must work for any input files. The files provided with this assignment are only one possible version, and must not be considered as the general case when writing your code.
g. To make sure that the requirments are very clear to you, Figure 7 given an image (partially) of a sample input file, and Figures 8, 9 & 10 show the output of this sample file in the 3 formats. These files are also provided with the assignment.

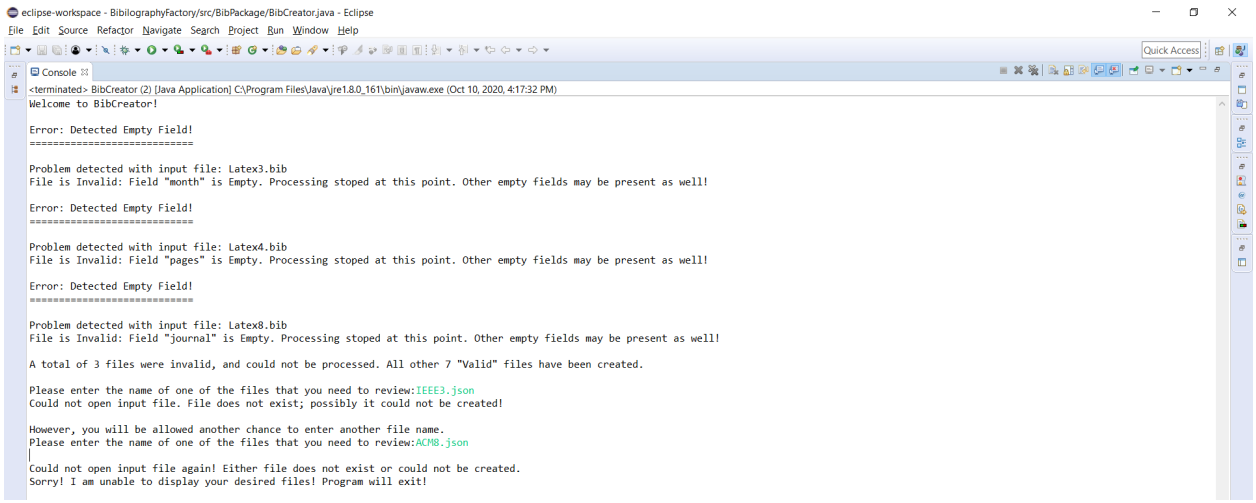

Figure 5. Example of Double-Fault for Displaying an Output File

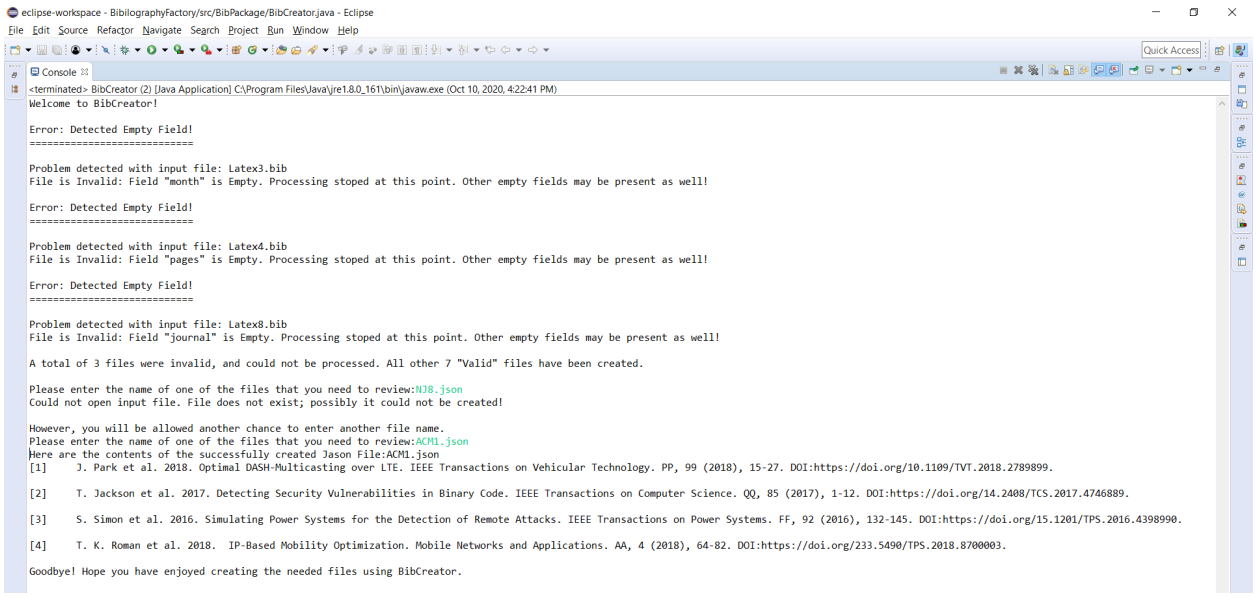

Figure 6. Example of Displaying the Contents of a Successfully Created Output File

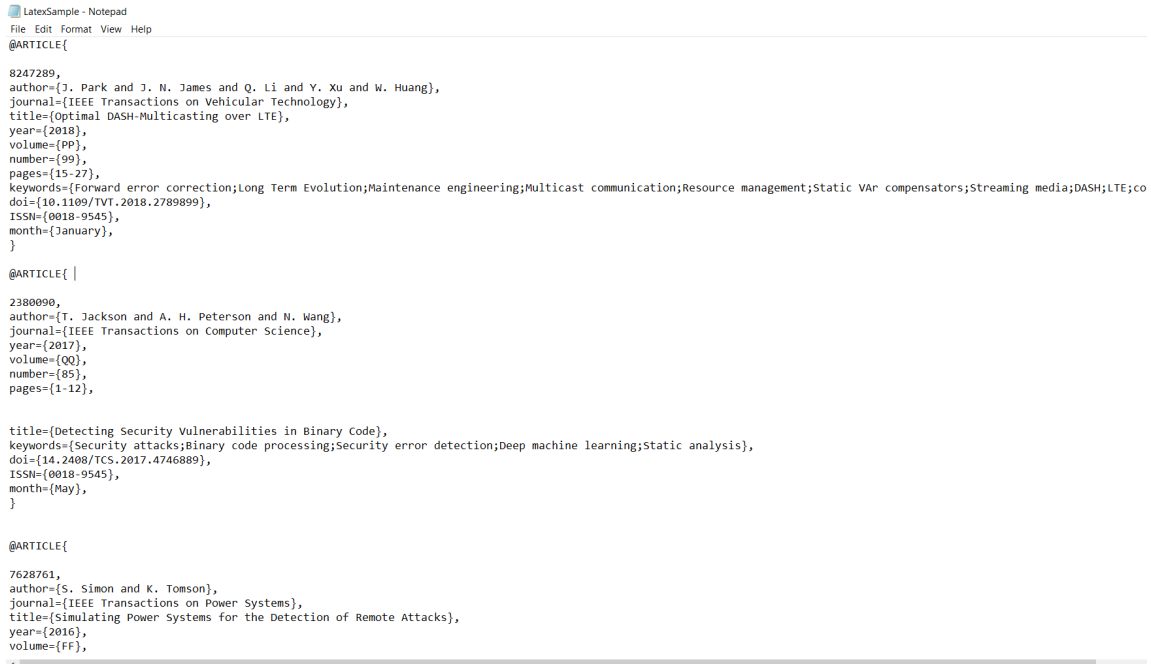

Figure 7. Example of a Sample Bib Input File – Partial Image

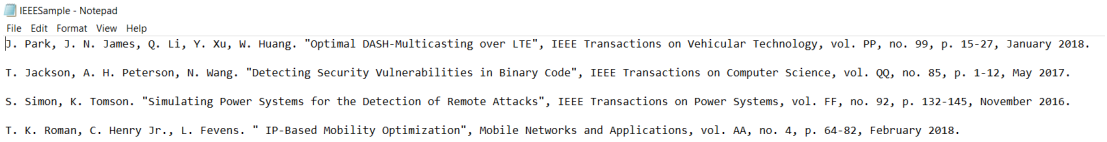

Figure 8. The Created IEEE File for the above Sample Bib File

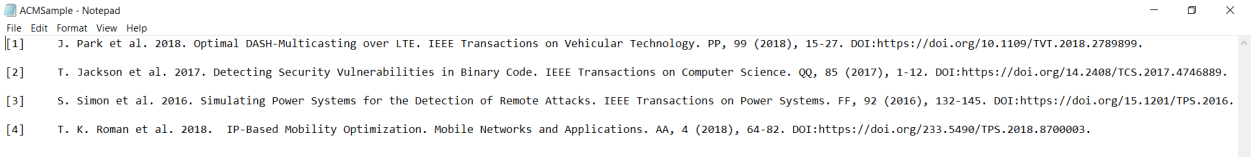

Figure 9. The Created ACM File for the above Sample Bib File

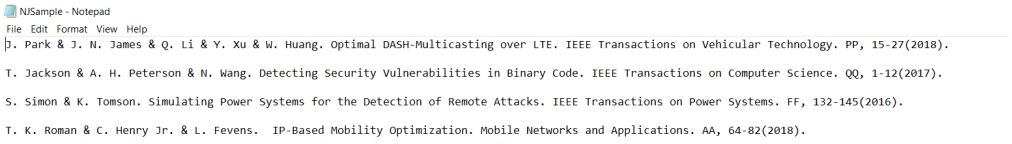

Figure 10. The Created Natural Journal File for the above Sample Bib File

Submitting Assignment 2

You need to submit Part I and Part II separately.
Part I:

- **Part I must be submitted individually (No groups are allowed) under the submission folder: Assignment 2 – Part I.**

Part II:

- **For Part II, a group of 2 (maximum) is allowed. No additional marks are given for working alone.**
- **If working alone, you need to zip (see below) and submit your zipped file under the submission folder: Assignment 1 - Part II.**
- **If working in a group, only one submission is to be made by either of the two members (do not submit twice). You need to zip (see below) and submit your zipped file, under the submission folder: Assignment 2 - Part II.**

Submission format: All assignment-related submissions must be adequately archived in a ZIP file using your ID(s) and last name(s) as file name. The submission itself must also contain your name(s) and student ID(s). Use your "official" name only - no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. If working in a group, the file name must include both IDs and last names.

IMPORTANT: For Part II of the assignment, a demo for about 5 to 10 minutes will take place with the marker. You (or **both** members if working in a group) **must** attend the demo and be able to explain their program to the marker. Different marks may be assigned to teammates based on this demo. The schedule of the demos will be determined and announced by the markers, and students must reserve a time slot for the demo (only one time-slot per group). **Now, please read very carefully:**

- **If you fail to demo, a zero mark is assigned regardless of your submission.**
- **If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a penalty of 50% will be applied.**
- **Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.**

Evaluation Criteria for Part II (10 points)

IMPORTANT: **Part I** must fully be submitted. Failure to submit that part will cost 50% of the total marks of the assignment!

Total	**10 pts**
JavaDoc documentations	1 pt
Task # 2, Task 3 & Task # 4	1 pt
Task # 5	4 pts
Task # 6	2 pts
Task # 7	1 pt
General Quality of the Assignment	1 pt