Instructors: Tuna Tuğcu & Cem Ersoy
TAs: Adalet Veyis Turgut & Selen Parlar
Contacts: adalet.turgut@boun.edu.tr & selen.parlar@boun.edu.tr

Due: **January 13, 2022, Thursday, 23.59**

1 Introduction

In this project, you are expected to create a multithreaded scientific search engine in $C/C++$ that queries the paper abstracts and summarizes the most relevant ones.

The project will be evaluated automatically in the Linux environment (Ubuntu Version 20.04.1) with either gcc/g++ or cc compiler (Version 9.3.0). Please follow all the requirements specified below. Your submissions will be compiled and tested via automatic scripts. Therefore, it is crucial that you follow the protocol (i.e. the rules and the specifications) defined in the project document. Failure in the compilation and/or execution is your responsibility. You should use the file names, parameters, etc. as mentioned in the project specifications.

2 Project Description

The *Abstractor* is a multithread-powered research tool for scientific literature. It focuses on delivering fast and accurate results for given queries. The program takes a query as an input, scans its file collection, finds the most relevant abstracts by measuring the similarities between abstracts and the given query, then retrieves convenient sentences from the selected abstracts.

3 Input & Output

Make sure that your final submission compiles and runs with these commands. We will run your code as follows:

```
make
./abstractor.out input_file_name.txt output_file_name.txt
```

3.1 Input

- The *abstractor.out* program takes two command line arguments:
  1. input_file_name.txt: Path of the input file
  2. output_file_name.txt: Path of the output file
- The input_file_name.txt file contains several parameters, which will be read line-by-line:

1. The first line contains values of T, A, and N, which are separated by spaces.
   T: Number of threads (other than the main thread) that will be created for the session
   A: Number of abstracts that will be scanned by the threads
   N: Number of abstracts that will be returned and summarized
2. The second line contains the words to query. An example query is provided below:
   virus pandemic global
3. From lines 3 to A+2, the names of the abstract files will be listed as follows.
   abstract_1.txt
   abstract_2.txt
   ...
   abstract_A.txt

- An input file consists of several lines of sentences that are pre-processed and separated by dots. An example abstract, which consists of two sentences, all in lowercase, and separated from punctuation, is provided below.

  the covid19 pandemic caused by sarscov2 remains a significant issue for global health economics and society . a wealth of data has been generated since its emergence in december 2019 and it is vital for clinicians to keep up with this data from across virus the world at a time of uncertainty and constantly evolving guidelines and clinical practice .

- The file structure for the evaluation is provided below. Please design your code and zipped file accordingly. We will provide you absolute paths, so you do not need to create directories.

```
STUDENT_ID.zip
src
    input_file_name.txt
    outout_file_name.txt
    Makefile
    abstractor.out
    < source code in c or cpp>
    < source code in c or cpp>
abstracts
    abstract_1.txt
    abstract_2.txt
    abstract_A.txt
```

3.2 Output

The *Abstractor* outputs the thread operation logs, scores, and resulting sentences to the given text file. An example output is provided in Table 4.

- Each thread outputs its name and operation as **"Thread *X* is calculating abstract_*Y*.txt"**.
- You should insert ### after each separate operation, i.e., after finishing threads jobs, listing result 1, result 2, and so on.

4 Thread Creation

Since text processing is a costly process and the texts we process are independent of each other, it is better to use parallelism in order to accelerate the *Abstractor* program. For that purpose, it is wise to use threads. Yet, the number of threads must be reasonable, so you should not create hundreds of them initially, and drain all system resources. You should create $\boldsymbol{T}$ number of threads and make them work till the end of the program.

Each thread must log its own name and the name of the file it is processing to the output file. You should name your threads alphabetically and start from A and continue with B, C, and so on.

You should evenly spread the jobs to the threads. So, some threads should not be idle while one thread does all the job. When all the abstracts are processed, all of the threads must be terminated gracefully. You need to use POSIX threads provided by Linux-based systems or MacOS for the implementation of this project in order to create a multithreaded environment (Windows is not allowed). To address the synchronization issues, you can use mutex and semaphores.

5 Similarity Measurement

In Natural Language Processing, we often need to estimate text similarity between text documents. There exist several text similarity metrics such as Cosine Similarity, Jaccard Similarity, and Euclidean Distance Measurement. All these text similarity metrics have different behavior.

In the *Abstractor* program you should use the Jaccard Similarity metric to determine the similarity between two texts. This metric finds how close the two texts are to each other in terms of the number of common words that exist over total words. Let's calculate the Jaccard Similarity of *Text1* and *Text2*:

1. Get two texts to compare:

   Text1: the war and the dance

   Text2: when the world was at war before we just kept dancing

2. Tokenize texts using space and find the unique words:

   WordSet1 = {the, war, and, dance}

   WordSet2 = {when, the, world, was, at, war, before, we, just, kept, dancing}

3. Find the intersecting words of two sets:

   Intersection_WS1_WS2 = {the, war}

4. Find the union of two sets:

   Union_WS1_WS2 = {the, war, and, dance, when, world, was, at, before, we, just, kept, dancing}

5. Calculate the Jaccard Similarity using the mathematical formula given below:

$$J(A, B) = \frac{\|A \cap B\|}{\|A \cup B\|} \tag{1}$$

$$J(Text1, Text2) = \frac{\|Intersection\_WS1\_WS2\|}{\|Union\_WS1\_WS2\|} = \frac{2}{13} = 0,1538 \tag{2}$$

6. Note that you are expected to output the Jaccard Similarity score with a precision 4.

6 Example Test Case

An example test case is provided below and also as an attachment to the description. (Please see the abstract_3.txt and abstract_6.txt from abstracts folder.) Table 1 shows an input file, which expects you to run your program with two threads and scan four abstract files (namely abstract_1.txt, abstract_3.txt, abstract_6.txt, and abstract_5.txt) and return the top two results.

2 4 2
virus pandemic global
abstract_1.txt
abstract_3.txt
abstract_6.txt
abstract_5.txt

Table 1: Sample input file for the *Abstractor*

Tables 2 and 3 show the content of files abstract_1 and abstract_5, respectively. Note that, text files consist of alphanumeric characters (all in lower case), spaces (of length 1), newline characters, and dots.

since its emergence in december 2019 corona virus disease 2019 covid19 has impacted several countries affecting more than 90 thousand patients and making it a global public threat . the routes of transmission are direct contact and droplet and possible aerosol transmissions . due to the unique nature of dentistry most dental procedures generate significant amounts of droplets and aerosols posing potential risks of infection transmission . understanding the significance of aerosol transmission and its implications in dentistry can facilitate the identification and correction of negligence in daily dental practice . in addition to the standard precautions some special precautions that should be implemented during an outbreak have been raised in this review .

Table 2: Sample text file, abstract_1.txt

the covid19 pandemic caused by sarscov2 remains a significant issue for global health economics and society . a wealth of data has been generated since its emergence in december 2019 and it is vital for clinicians to keep up with this data from across the world at a time of uncertainty and constantly evolving guidelines and clinical practice .

Table 3: Sample text file, abstract_5.txt

Using the formula given in Equation 1, you are expected to find the two most similar abstracts to the given query and return the similarity score. Besides that, you should perform an extractive summarization, i.e., you should extract the sentences from the corresponding abstracts that have at least one common word with the given query. Table 4 shows the output file for the given input and abstract files. In this example, abstract_5 is the most similar one to the query of "virus pandemic global". And the sentence "the covid19 pandemic caused by sarscov2 remains a significant issue for global health economics and society ." is returned since it is the only sentence that contains two words from the given query with the score of 0.0400. Since we will grade your projects automatically, please pay attention to the headers such as Result N:, File:, Score:, Summary:, and ###.

While implementing these functionalities, it is important to pay attention to the common resources and race conditions. There might be several possible orderings of threads depending on the scheduling used by your system; we will be checking the output files. However, it is important that you are able to spread the work evenly.

```
Thread A is calculating abstract_1.txt
Thread B is calculating abstract_3.txt
Thread A is calculating abstract_6.txt
Thread B is calculating abstract_5.txt
###
Result 1:
File: abstract_5.txt
Score: 0.0400
Summary: the covid19 pandemic caused
by sarscov2 remains a significant issue
for global health economics and society .
###
Result 2:
File: abstract_1.txt
Score: 0.0238
Summary: since its emergence in december
2019 corona virus disease 2019 covid19
has impacted several countries affecting
more than 90 thousand patients and making
it a global public threat .
###
```

Table 4: Sample output

7 Report & Grading

You are expected to submit a well-documented code. You need to explain the parameters, variables, and functions used in the code. You also need to provide author information, the main idea of the project, and a conclusion. Corresponding points are **tentatively** specified in parentheses.

1. **Multithread handling:** Your program should run tasks using multiple threads. *(40 pts)*
2. **Test cases:** Your program will be tested with various test cases. *(30 pts)*
3. **Code&Comment:** *15*
4. **Auto-runnable submission:** Submit code that runs smoothly with the specified commands in the Linux environment. Each re-submission deduces 5 points. *(10 pts)*
5. **Score calculation:** Implement the simple text similarity algorithm. *(5 pts)*

8 Submission

Submissions will be through Moodle. Submit a single .zip file named with your student ID (e.g. 2019400XXX.zip) that matches the specified structure. Your zipped project should directly contain the required file structure, not the folder that contains the files structure. Please pay attention to the file naming and the content you should be printing while preparing the output file.

Note that your project will be inspected for plagiarism with previous years' materials as well as this year's. Any sign of **plagiarism** will be **penalized**.