## Project 5

You are going to write a complete C program which implements the following functionality:

- Your program will read the following files:

```
language_1.txt
language_2.txt
language_3.txt
language_4.txt
language_5.txt
language_x.txt
```

- Each file contains text in a specific language. All files contain only english lowercase characters and whitespace. Text files will include the following characters:

```
'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n'
'o' 'p' 'q' 'r' 's' 't' 'u' 'v' 'w' 'x' 'y' 'z' ' '
```

- Your program will evaluate the dissimilarity scores of language pairs:

```
(language_x, language_1)
(language_x, language_2)
(language_x, language_3)
(language_x, language_4)
(language_x, language_5)
```

- First of all, calculate bi-gram frequencies for each language. A bi-gram is defined as follows: For a given sequence, each unique pairing of successive letters is a bi-gram. For example: for the word `" adana "` bi-grams are defined to be `" a", "ad", "da", "an", "na", "a "`. Beware: If there is a space before or after a character you will still be dealing with bi-grams. Each bi-gram has exactly two elements which are either characters or space. In order to calculate the frequency of a particular bi-gram(lets say bi-gram `"ad"`) you have to count all the bi-grams in a given text and for this bi-gram calculate the ratio `(# of "ad")/(total # of all bi-grams)`
- Given all the frequencies, dissimilarity score is calculated as follows:

$$dissimilarity(language_a, language_b) = \sum_i |f_a^i - f_b^i| \tag{1}$$

- Here $f_a^i$ represents the frequency of $i^{th}$ bi-gram for the language $a$. If $c_a^i$ is the count of $i^{th}$ bi-gram in $language_a$, then;

$$f_a^i = c_a^i / (\sum_j c_a^j) \quad (2)$$

- After evaluating dissimilarities, your program will print all the dissimilarity values. Print:

```
dissimilarity(language_x, language_1)
dissimilarity(language_x, language_2)
dissimilarity(language_x, language_3)
dissimilarity(language_x, language_4)
dissimilarity(language_x, language_5)
```

## Remarks:

- text files can include multiple concatenating whitespace. For example:

```
Here          we    are using a user defined    recursive
```

- Two adjacent whitespace do not create a bi-gram.
- Input files can be multi-line text files.
- There isn't any limit on the size of input files. Your program should work regardless of the size of the input.

## Hints:

- Bi-gram types do not depend on the input file. There are finite number of possibilities. Given all the lowercase english characters and a space, you can generate all the possible bi-grams.
- Do not try to store all the content of the file in the memory. Counting is possible without storing all of the text.
- You don't need to parse words.

## Turn in:

A complete C program <Project5.c> which can be compiled using the following command:

```
gcc -std=c99 Project5.c -o Project5.c
```

If your program requires additional compile and link options, state that requirement at beginning of your source code as a comment.

## Caution:

- Read and apply "Assignment Submission Rules and Other Related Information" document which available on the class e-learning system.
- You may or may not get partial credit depending on how you structured or documented your code.