Plagiarism Checker

In this assignment, you are required to implement a plagiarism detection tool in C++. The tool will compare the textual similarity between two documents by processing the text to remove irrelevant information and then apply Cosine Similarity to calculate a similarity score.

Files Detail:

You have following text files (input.txt, document1.txt & document2.txt):

1. input.txt:

    a. The first row of the file contains Stop Words. Stop Words are lists of words which you don't want to include in analysis. For example, you might want to make a word list excluding common function words like *the, of, was, is, it*.
    Note: Words are space separated. And stop words must be in lower case.
    b. The second row of the file contains number of documents. For example, "**2**" in this case.
    c. Followed by relative paths of documents. **(Place the .txt and .cpp files in same folder)**

    Example:

    Stop Words: the of also was a an is to it too
    2 document1.txt
    document2.txt

2. Dataset consists of one or more documents. **There can be more than two documents.** (In our example there are two documents).
    a. document1.txt: "John likes to , watch movies. Mary: likes movies too."
    b. document2.txt: "Mary also' likes # to& watch football games."

Steps:

Process (or Preprocess, to be technically correct) the documents as follows:

Step 1:

a. You will read Stop Words form *input.txt* file
b. Read one or more documents whose path is mentioned in *input.txt* file (For example, *document1.txt* & *document2.txt* file).

Step 02:

Remove all the punctuations marks **(. ~ ! @ # $ % ^ & * ( ) _ + = " ; : / ? > , <).** Including **'\n' (newline) & extra spaces.**

Note: Remember

1) Output must only contain **alphabets and space characters.**

2) **" 's " is not be included in student and evaluation testcases.**

3) There must be **single space between two words. Output:** "John likes to watch movies Mary likes movies too" **Step 03:**

Convert all upper-case letters to lower case.

Output: "john likes to watch movies mary likes movies too" **Step 04:**

Remove stop words

Output: "john likes watch movies mary likes movies"

"mary likes watch football games" **Step 05:**

Now generate frequencies.

Term	Document1	Document2
john	1	0
likes	2	1
watch	1	1
movies	2	0
mary	1	1
football	0	1
games	0	1

Note: unique words must be extracted according to the document's order i-e order should be maintained (first unique word should be "john" then "likes", so on).

Step 06:

Using the word frequency arrays generated, calculate the Cosine Similarity between two documents. This similarity score will quantify how similar the two documents are, with a score ranging from 0% (completely dissimilar) to 100% (identical).

Method for Calculating Cosine Similarity:

1. **Generate Vectors**:
    - First, create a array for each document based on the frequency. Each array should contain the frequency of each unique word present in both documents.
    - If a word is present in one document but not the other, its frequency in the corresponding index should be set to 0.
2. **Dot Product Calculation**: Compute the dot product of the two arrays. The dot product is calculated as the sum of the products of corresponding elements from both arrays.

$$\text{Dot Product} = \sum_{i=1}^{n} A_i \times B_i$$

3. **Magnitude Calculation**: Calculate the magnitude (length) of each array. The magnitude of a array is the square root of the sum of the squares of its components.

$$\text{Magnitude of A} = \sqrt{\sum_{i=1}^{n} A_i^2}$$

$$\text{Magnitude of B} = \sqrt{\sum_{i=1}^{n} B_i^2}$$

4. **Cosine Similarity Calculation**: Finally, calculate the cosine similarity by dividing the dot product of the two arrays by the product of their magnitudes.

Example:

Suppose you have two documents with the following frequency arrays:

Document 1 array: [1, 2, 1, 2, 1, 0, 0]

Document 2 array: [0, 1, 1, 0, 1, 1, 1] 1.

Dot Product:

$$\text{Dot Product} = (1 \times 0) + (2 \times 1) + (1 \times 1) + (2 \times 0) + (1 \times 1) + (0 \times 1) + (0 \times 1)$$

$$\text{Dot Product} = 0 + 2 + 1 + 0 + 1 + 0 + 0 = 4$$

2. **Magnitude of Vectors**:

Magnitude of Document 1:

$$\text{Magnitude of Document 1} = \sqrt{1^2 + 2^2 + 1^2 + 2^2 + 1^2 + 0^2 + 0^2}$$

$$\text{Magnitude of Document 1} = \sqrt{1 + 4 + 1 + 4 + 1 + 0 + 0} = \sqrt{11}$$

$$\text{Magnitude of Document 1} \approx 3.32$$

Magnitude of Document 2:

$$\text{Magnitude of Document 2} = \sqrt{0^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2 + 1^2}$$

$$\text{Magnitude of Document 2} = \sqrt{0 + 1 + 1 + 0 + 1 + 1 + 1} = \sqrt{5}$$

$$\text{Magnitude of Document 2} \approx 2.24$$

3. **Cosine Similarity**:

$$\text{Cosine Similarity} = \frac{\text{Dot Product}}{\text{Magnitude of Document 1} \times \text{Magnitude of Document 2}}$$

$$\text{Cosine Similarity} = \frac{4}{3.32 \times 2.24} \approx \frac{4}{7.44} \approx 0.54$$

The cosine similarity between Document 1 and Document 2 is approximately **0.54**, which suggests a moderate level of similarity between the two documents based on the terms provided.

You need to return the answer in percentage with 2-digit precision. For above, case the answer will be **53.94%.**

Function Prototypes:

1. **void readInput (const char* pathofInputFile) pathofInputFile** indicates path of input.txt file.

2. **int getNumberOfDocument ()**
   This function will return number of documents.

3. **char * getText (int documentNumber)**

This function will return document text as char array (char*) according to document number. Note: *getNumberOfDocument & getText functions will be used for testing purposes.*

4. **void removePunctuationMarks ()**

   This function will remove punctuation marks and make one long sentence of whole document with 1 space. As shown in **Step 02.**

5. **void convertUpperToLowerCase ()**

   This function will convert upper case letters to lower case letters according to **Step 03**

6. **void removeStopWords ()**

   This function will remove stop words according to **Step 04**

7. **void generateFrequencies (char**& uniqueWords, int& uniqueCount, int**& documentFrequency)**

   This function will identify unique words and count frequencies in all documents for unique words. **Step 05**

8. **int getFrequency (char* word, int doucmentNum) // -1 if not found**

   This will return the frequency of specific word in specific document.

9. **void calculateAllCosineSimilarities (double**& similarities, int** documentFrequency)**

   This function will calculate all the Cosine Similarities and save them in array. Like following example for 3 documents. Table will be symmetric. **similarityIn (1,2) = similarityIn (2,1) = 52.10**

	Document 1	Document 2	Document 3
Document 1	100	52.10	90.92
Document 2	52.10	100	10.10
Document 3	90.92	10.10	100

10. **double similarityIn (int doucmentNum1, int documentNum2)**

    Return similarity of any 2 documents.

---------- **Happy Coding** ----------