CS 218 DATA STRUCTURES
ASSIGNMENT 5
SECTION B, C, D and G
Fall 2020

DUE: Dec 18, 2020
TO SUBMIT: Documented and well written structured code in C++ on classroom. Undocumented code will be assigned a zero.

PROBLEM BACKGROUND
We want to design a small search engine that can help its users to retrieve documents related to the search query from a collection of documents as done in Assignment 1&2.
In assignment 1&2, the unique terms were stored in an unordered array/linked list. The search operation is therefore O(n) in the worst case. If we replace the array/list with a balanced BST, then we can search these terms in O(lgn) time in the worst case. Also if we replace the array/linked list with a hash table then we can search these terms inO(1) on the average. You task is to redo the assignment1&2 but now with Hash Tables. Use chaining as the collision resolution strategy. You can use hash function of your choice that you think is most suitable for this application.

IMPLEMENTATION
Your task is to design a small search engine that can perform following functionality:
Create_Index: Given the collection of documents, tokenize words in each document, compute their term frequencies and create the index.
Search_Documents: Given the query word(s), output a ranked list of related documents as done in assignment 1&2.
Add_Doc_to_Index: Given a new document (DOC ID and the text), tokenize its words, compute their term frequencies. If a word is already present in the index, add the Doc ID and computed term frequency at the end of the corresponding list. Otherwise add the new word in the Hash Table.

Important Note: For simplicity, you can assume that all the documents and query do not contain any stop word and all words are separated by white space.

IMPORTANT CLASSES
You have to implement the following classes
Class List
Use the class List that you designed in assignment 1&2
Class Hash Table
Implement Hash Table class and use List class to implement chaining. The hash table class must have data members Table size and maxsize. Where Table is a dynamic array of type List whose size is max size. Size is the total number of elements in the hash table. Implement all the required operations including grow functions. Grow function must be called whenever the load factor of the table exceeds 2. Grow function must allocate a new table of double the current maxsize, rehash all the previous elements in the new table and delete the old table.
Class Doc_Info
This class must contain two data members DocID and term frequency of a particular term
Class Term_Info
This must contain a key term and a list of Doc_Info as its data members.

Class Search_Engine
This must contain a Hash Table **Index** of type List<Term_Info>, Following is the list of required functions:

Create_Index: This function takes an array ***Docs*** of type strings/char* that contains the file names and an integer ***n***. Here n is the size of array ***Docs***. For each file in ***Docs***: call **Add_Doc_to_Index**

Search_Documents: Given the query word(s), output a ranked list of related documents as done in assignment 1&2.

Add_Doc_to_Index: Given a file name: open it, read it and tokenize words on white space. Also, compute the term frequency of each unique word in the file. For each unique word, if the word is already present in the index, add the Doc ID and computed term frequency at the end of the corresponding Doc_Info list. Otherwise add the new word in the Index.

Add any function that you find important to implement above mentioned functions.
For simplicity you can declare Doc_info and Term_info as friends of Search_Engine.