The term Term Frequency (TF) is used in the field of information retrieval to indicate the frequency (number of occurrences) of a word in a document. Search engines identify the relevance of a particular document to a query based in part on the TF of the words in the document.

In this exercise, the program will analyze a text file (document) and build the list of meaningful words (other than stopwords) and its frequency (which appear above a certain threshold of times).

stopwords are very common words that appear in almost every document and do not have a substantive meaning such as 'the', 'and' and the like.

The following interface is a library interface that maintains a data structure of a linked list of words:

// wordlist.h

#ifndef WORD\_LIST\_H

#define WORD\_LIST\_H

typedef struct {

char \*data;

int occurrences;

}Word;

typedef struct wordnode {

Word wordData;

struct wordnode \*next;

}WordNode;

typedef struct {

WordNode\* head;

}WordList;

WordList\* createList();

void addSorted(WordList\* list, char\* data);

void removeList(WordList\* from, WordList\* toRemove);

void printList(WordList\* list);

void destroyList(WordList\* list);

#endif

The interface defines the following three structures:

Word

Contains data on a word. The structure contains the fields:

data - char \* type, contains a pointer to the word string itself.

occurrences - type int, contains the TF, the number of instances of data in the text.

The WordNode structure

Defines an organ in a linked list of words. The structure contains the fields:

wordData - Word type, contains the word that the organ saves in the list.

next - WordNode \*, contains the address of the next member in the list (the last member in the list will contain NULL in this field).

The WordList structure

Defines the data structure of a linked list of words. Contains the field -

head - WordNode \*, points to the top of the list (the first member of the list). When the list is empty its value will be NULL.

The interface also defines the five functions for performing operations on a list:

WordList \* createList () - The function creates an empty list by allocating dynamic memory to the WordList structure and placing NULL in the head field. The function will return the pointer to the created list. If the function failed to create the list for some reason, the NULL value will be returned.

void addSorted (WordList \* list, char \* data) - The function gets a pointer to a linked list and a string representing a word.

If the word data does not already exist in the list, the function will create a new member in the list with the requested word, and the number of occurrences 1. The function will add the word to the list, sorted. Note that in this case a dynamic memory must be created for the string!

If the word data already exists in the list, the function will add 1 to the number of occurrences of the word.

void removeList (WordList \* from, char \* toRemove) - The function gets a pointer to a linked list and a string representing the name (full path) of a file.

The function will remove from the list from all the words that appear in the toRemove file.

Note that removing a word means removing a word from the list (after releasing the memory assigned to it) and not reducing the number of instances by 1.

void printList (WordList \* list) - The function receives a pointer to a linked list. The function will print the list members in the following format: each word will appear in a separate line, in which the word itself will appear, followed by a comma followed by the number of occurrences.

void destroyList (WordList \* list) - The function receives a pointer to a linked list. The function erases the entire list from memory. Note that the function should free up all the memory allocated for the list members and each memory member allocated for the string. In addition to releasing the voter's memory to the list itself.

1. Write a file named wordlist.c that implements the described interface. The file will contain the description of the interface by the include:

#include "wordlist.h"

And in addition cumin the realization of the five functions described above.

2. Write a plan that implements file analysis. The program was written in a file called analyzetext.c.

The main function of the program will perform the following:

• Read a text file and save its words in a list of common words.

• Print the list received

• Delete words whose number of occurrences is less than a certain threshold

• Print the list after deleting

• Read a file containing common words and delete them from the list

• Print the list after the second delete

The constant MIN\_OCCURENCES should be set to 2 in the file using #define.

The program will receive two arguments through the command line. The first argument is the (full) path of the document file and the second argument is the (full) path of the common word file.

An example of a document file (document.txt) and an example of a common word file containing one word per line (stopwords.txt) can be found in the task set.