# ECE 275 Assignment 2

**DUE DATE:** Thursday, October 9, 11:59PM

In this assignment, you will create an application that analyzes the frequency of the occurrence of words within two inputs files, a start file and an end file. The application will further analyze the change in frequency of specific words from the start file to end file, to determine the increase or decrease in the word frequencies. Finally, the program will output (using standard output) the five words that have increased the most and the five words that have decreased the most from the first input file to the second file.

Note: This program is motivated by http://whatstrending.com that monitors Twitter feeds to see what topics are increasing and decreasing in popularity.

The assignment name for this assignment is: **trending**

## Command-line Arguments

Your program must be capable of utilizing a command-line argument to specify the input files.

trending startFile endFile

Your program must ensure the user has correctly provided the required command-line arguments and display a usage statement if the provided arguments are incorrect.

## Input Text Files

For this assignment, the input text file will only consist of uppercase and lowercase characters ('a' to 'z', 'A' to 'Z') and numeric characters ('0' to '9'). Each word within the text file will be separated by at least one whitespace character (' ', '\t', '\n').

Please note that for this assignment you ***DO NOT*** need to check that input file matches this format. Instead, you may simply ***assume*** that it will. In other words, your program can just read in strings from the input file  (separated by whitespace) and assume the strings are in the correct format.

**Note**: Text files on Windows based computers use a carriage return ('\r') and newline ('\n') at the end of each line. On Unix machines (such as the ece3 server), only the newline is used. For testing your program, you should utilize Unix formatted text files without the carriage return. If you edit your files using a Windows based program (such as Notepad), you may want to familiarize yourself with the dos2unix command available on the ece3 server.

## Doubly-Linked List Data Structure

For keeping track of the individual words and how many times they are used within the input textile, you must implement a doubly-linked list. The following struct and typedef definitions ***must*** be used for the linked-list. ***DO NOT*** make modifications to these data structures in your assignment.

typedef struct DListNode\_struct {

char \*word;

unsigned long startCount;

unsigned long endCount;

struct DListNode\_struct \*next;

struct DListNode\_struct \*prev;

} DListNode;

typedef struct DList\_struct {

int size;

DListNode \*head;

DListNode \*tail;

} DList;

The following functions ***must*** be implemented for the doubly-linked list using the provided function declarations. ***DO NOT*** make modifications to the function names, return types, parameter names, or parameters types.

void DListInit(DList\* list);

void DListDestroy(DList\* list);

bool DListInsertAfter(DList\* list, DListNode\* currNode, DListNode\* newNode);

bool DListInsertBefore(DList\* list, DListNode\* currNode, DListNode\* newNode);

DListNode\* DListSearch(DList\* list, char\* key);

bool DListRemove(DList\* list, DListNode\* currNode);

The functionality specific to these data structures should be implemented within their own set of C source and header files named dlist.h and dlist.c. ***You may add more functions to these files, but you may not change the above function signatures in any way***.

**Note**: If your program does not use a doubly-linked list as indicated in the program assignment, you will receive 0 points for the assignment.

**Hint**: The bool data type is not defined in C, but it wasn’t defined in C during Assignment 1 either. Check out the template we provided for Assignment 1 to recall how to define bool as a type (you can even use the same file you used in the last assignment).

**Pro tip:** You can infer the behavior of these functions if you check out the chapters on Containers in the Zyante text. One additional recommendation is to insert at the head of the list when calling DListInsertAfter and DListInsertBefore if currNode is NULL;

## Word Counting

As each word is read from the input text file, your program should keep track of the number of times each word is utilized. While the input file may contain both uppercase and lowercase characters, the identification of unique words is case insensitive. For example, "Party", "party", and "PARty" are all considered the same word.

**Hint**: Use tolower()

Your program will need to keep track of the number of times each word is found within both the start file and the end file.

**Pro tip**: Create files wordcounting.h and wordcounting.c to incorporate logic to read in and count words. Do not put all this function in main, or you will receive deductions for bad design.

## What's Trending

The program should determine the top five words that are Trending Up and the top five words that are Trending Down. The five words trending up are those words that have increased the most in frequency from the start file to the end file. The five words trending down are those words that have decreased the most in frequency from the start file to the end file.

The trending up and trending down words should be reported using the following format. If ties occur within these lists, the ordering does not matter. The value in parentheses is the difference between the end count and the start count for words reported.

Trending Up:

believe (+100)

choose (+90)

i (+80)

programmed (+40)

to (+2)

Trending Down:

programming (-1024)

in (-800)

c (-456)

is (-7)

hard (-1)

**Pro tip**: Create files trending.h and trending.c to incorporate the logic specific to the trending calculation. Do not put all this logic in main, or you will receive deductions for bad design.