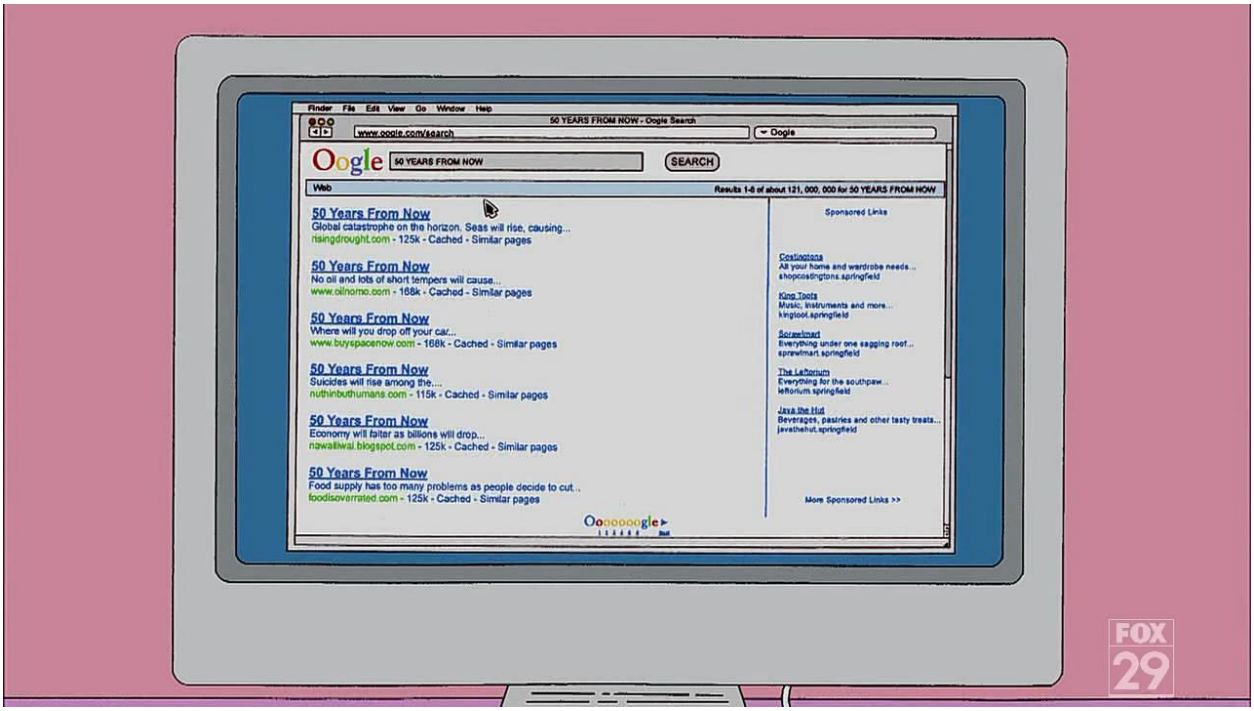
# Computer Engineering Case Study
# Concordance of a Text File





**Dhiyaa Al Jorf**

**da2863**


ENGR-UH 1000 Computer Programming for Engineers
Assignment 3

## Step 1 – Problem Identification & Statement:

As a first step in intelligent information gathering, the objective is to develop a program that can return the concordance of a text file regardless of the size of said file. The concordance of a text file is an alphabetical list of the unique words in the text file along with the number of times each word occurs (frequency). The program should both output the concordance and save it into a text file.

## Step 2 – Gathering Information & Input/Output Description:

### Definition:

The concordance of a block of text is a sorted list of unique words in said text and their frequency of occurrence.

### Applications:

Getting the concordance of a block of text can be very helpful in multiple fields including:

- Determining the main topics present in a given block of text. The most frequently occurring words (aside from articles, prepositions, etc.) often gives the main topic of a text.
- Searching on the web. As an extension of the previous points, search engines can use the concordance to find content that matches the search criteria.
- Finding similarities between multiple pieces of literature. The commonalities between the concordances of two files might indicate a similarity between the texts.

### Program Specifications:

The software must meet the following specifications:

- The program prompts the user for the name of the input text file where the text is stored. The program must print an error message in case errors occur while opening the files. The program must read words and store them into an array of strings. Punctuation characters and all nonalphabetical characters must be ignored and used as delimiters. All alphabetical characters must be converted to lower case characters to eliminate case sensitivity. The program only needs to handle ASCII characters (not all Unicode characters).
- The program must sort the words and search for the unique ones. It should also count the number of occurrences of these words in the list. The list of unique words and the frequency will be stored in dynamic arrays. Selection sort[1] will be utilized in this program to sort the words alphabetically.

[1] Selection sort is a sorting algorithm that works by going through a list of words and repeatedly finding the minimum (if the list is to be ordered ascendingly) and placing it at the beginning. For a list of n elements, the algorithm performs n-1 passes and runs at $O(n^2)$ time. Below is a visual example of selection sort at work:

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 5 | 87 | 32 | 14 | 22 | Unsorted Data |
| 5 | 10 | 87 | 32 | 14 | 22 | First Iteration |
| 5 | 10 | 87 | 32 | 14 | 22 | Second Iteration |
| 5 | 10 | 14 | 32 | 87 | 22 | Third Iteration |
| 5 | 10 | 14 | 22 | 87 | 32 | Fourth Iteration |
| 5 | 10 | 14 | 22 | 32 | 87 | Fifth Iteration |
| 5 | 10 | 14 | 22 | 32 | 87 | Sorted Data |

- The program saves the concordance list (list of unique words), along with the frequency of occurrence, in a data file that the user is prompted to provide. The program must print a confirmation message on the output screen once the data is stored in the file.
- The program must print the concordance list on the output screen.

**String Tokenization:**
The process of string tokenization is the splitting or breaking down of a string into several parts, be it phrases, words, characters, etc. according to set delimiters. For example, tokenizing a string that stores the phrase "My name is Dhiyaa" into words would produce the strings "My", "name", "is", "Dhiyaa", each of which is called a token.[2]

Our program must perform string tokenization on the incoming strings (the strings from the user-defined input file). As previously mentioned, these strings must be split into tokens according to the following rules:

- Alphabetical characters must be transformed to their lowercase version (e.g. "Firas is MY FrIeNd" must be tokenized into "firas", "is", "my", "friend")
- Non-alphabetical characters must be treated as delimiters, including apostrophes in contractions (e.g. "isn't" -> "isn" and "t", " 'I am' ^^NeA2ring__my limit :')" -> "i", "am", "near", "ing", "my", "limit")

## Input Output Description:

**Input:**

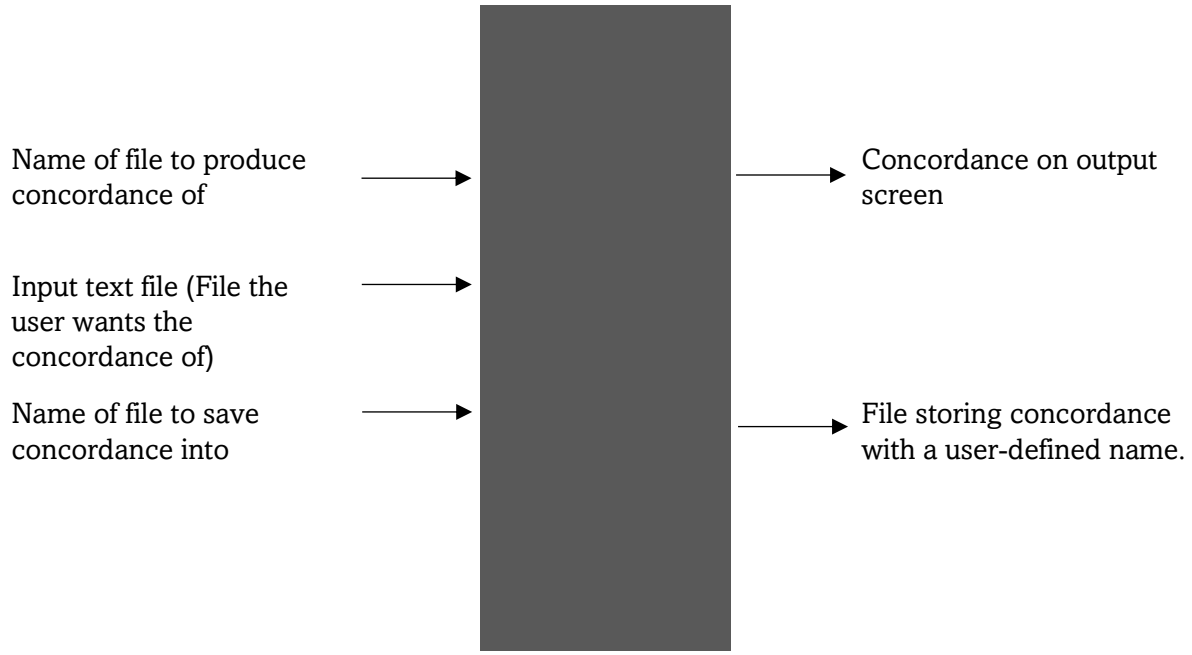
- Name of file to produce concordance of
- Input text file (File the user wants the concordance of)
- Name of file to save concordance into

**Output:**

- Concordance displayed on the output screen
- File storing the concordance with the user-defined name

[2] Debnath, Manoj. "Exploring The Java String Tokenizer | Developer.Com." *Developer.com*. N.p., 2021. Web. 14 Nov. 2021 . < https://www.developer.com/database/exploring-the-java-string-tokenizer/ >.

**I/O Diagram:**




## Step 3 – Test Cases & Algorithms:

### Test Cases:

All text used to test the code will be my own to avoid plagiarism.

**<u>Case 1 - Input file not present:</u>**

In case the user-defined input file doesn't exist, the program must print an error message and exit the program.

**<u>Case 2 – Small Paragraph:</u>**

Test Text:
"
This text has a lot of repeated words.
For example:
The word "repeated" is repeated 6 times in this text...
The word "text" is repeated 6 times in this text...
The word "the" is repeated 5 times in this text...
The word "this" is repeated 6 times in this text...
"

Expected Output:

The program should prompt the user to enter the name of the input file, then display the concordance of the file, and after that ask the user to enter the saving destination. Below is the expected concordance.

| WORD: | FREQUENCY |
|---|---|
| a | 1 |
| example | 1 |
| for | 1 |
| has | 1 |
| in | 4 |
| is | 4 |
| lot | 2 |
| of | 1 |
| repeated | 6 |
| text | 6 |
| the | 5 |
| this | 6 |
| times | 4 |
| words | 4 |

**Case 3 – Large Block of Text:**

Test Text:

"Before generating the file, the input in the configuration file must be validated according to the following constraints:

Wind Configuration Constraints:
- Average wind speed must non-negative and greater than the gust speed to avoid having (average speed - gust value) negative.
- Gust speed must be non-negative as speed can never be negative.
- The duration and step size must be non-zero positive for the data to sample different times.

Storm Configuration Constraints:
- The storm probability must be non-negative (0 probability should give us no storms) and written in decimal form (40% -> 0.4)
- Storm amplitude must be positive (min and max both should be positive) and the max amplitude must be greater than the minimum.
- Storm duration must be positive (min and max both should be positive) and the max duration must be greater than the minimum.

Microburst Configuration Constraints:
- The microburst probability must be non-negative (0 probability should give us no storms) and written in decimal form (40% -> 0.4)
- Microburst amplitude must be positive (min and max both should be positive) and the max amplitude must be greater than the minimum.

- Microburst duration must be positive (min and max both should be positive) and the max duration must be greater than the minimum. ”

Expected Output:
The user should be prompted to enter the names of the input and output file, the program must print the concordance on the output screen and save it in the user-defined file, and the concordance must contain exclusively lower-case characters with their correct frequency. The text is too long to perform concordance on it by hand and thus an online concordance generator will be used to verify that the correct frequencies have been input.

**<u>Case 4 – Text Containing Lots of Non-Alphabetical Characters:</u>**

Test Text:
“
Hello there! My name is Dhiyaa-Al-Jorf…
The time currently is 5:35 p.m.
Hello%%$#There2321  “How” -Are—You doing
The previous line should be split into the following words:
hello how are you doing
”

Expected Output:
The user should be prompted to enter the names of the input and output file, the program must print the concordance on the output screen and save it in the user-defined file, and the concordance must contain exclusively lower-case characters with all non-alphabetical ASCII characters treated as delimiters.

| WORD: | FREQUENCY |
|---|---|
| al | 1 |
| are | 2 |
| be | 1 |
| currently | 1 |
| dhiyaa | 1 |
| doing | 2 |
| following | 1 |
| hello | 3 |
| how | 2 |
| into | 1 |
| is | 2 |
| jorf | 1 |
| line | 1 |
| m | 1 |
| my | 1 |
| name | 1 |
| p | 1 |
| previous | 1 |
| should | 1 |

| | |
|---|---|
| split | 1 |
| the | 3 |
| there | 2 |
| time | 1 |
| words | 1 |
| you | 2 |

**Case 5 – Empty File:**

If the input file is empty, the program is expected to return an empty list of words and frequencies.

## Algorithm Design

```
readFile(ptrArr, reference to size)
        Print "Enter the file name (include file extension): ", Newline
        Read the next line into fileName
        Create an input file stream reader
        Open fileName into reader in the input more
        If reader fails to open the file
                PrintError "ERROR: Failed to open file . . .", Newline
                Exit with the code -1
        Create a string str
        Repeat While (Reading the next string from reader into str doesn't fail)
                trim(str, size, ptrArr)
        Close reader

add(reference to ptrArr, reference to size, value)
        Allocate memory in the heap for an array of string pointers of size size + 1 and save the
        address of the first element in tempPtr
        Assign 0 to i
        Repeat While (i is less than size)
                Assign ptrArr[i] to tempPtr[i]
                Increment i
        Allocate memory in the heap for a string storing value and save its address into
        tempPtr[size]
        Deallocate the memory pointed to by ptrArr
        Assign tempPtr to ptrArr
        Increment size

trim(str, reference to size, reference to ptrArr)
        Assign 0 to start
        Assign 0 to i
        Repeat While(i is less than the length of str)
                If (str[i] is non alphabetical OR (str[i] is alphabetical and i = (length of str - 1)))
                        If (str[i] is alphabetical)
                                Assign the lower case version of str[i] to str[i]
                                Increment i
```

```
                Allocate enough memory in the heap for an array of characters of size (i+1-
                start) and assign the address of the first element to res
                Assign 0 to j
                Repeat While(j is less than i – start)
                        Assign str[j+start] to res[j]
                        Increment j
                Assign '\0' to res[i – start]
                Assign res to the string result
                Deallocate all the memory pointed to by res
                Assign i + 1 to start
                If (result is not equal to "")
                        add(ptrArr, size, result)
        Else
                Assign the lower case version of str[i] to str[i]
        Increment i

sort(reference to ptrArr, size)
    Assign 0 to i
    Repeat While (i is less than size -1)
        Assign i to minIndex
        Assign (i+1) to j
        Repeat While (j is less than size)
            If (The value pointed to by ptrArr[minIndex] is greater than the value
            pointed to by ptrArr[j])
                Assign j to minIndex
            Increment j
        swap(ptrArr[minIndex], ptrArr[i])
        Increment i

swap(reference to x, reference to y)
    Assign x to temp
    Assign y to x
    Assign temp to y

concordance(ptrArr, reference to ptrFreq, reference to ptrUnique, size, reference to sizeUnique)
    Assign 0 to i
    Assign 0 to j
    Repeat While (i is less than size)
        add(ptrUnique, sizeUnique, value stored in ptrArr[i])
        Decrement sizeUnique
        Assign 1 to frequency
        Increment i
        Repeat While ((i is less than size) AND (the value pointed to by ptrUnique[j] is equal
        to the value pointed to by ptrArr[i]))
            Increment frequency
            Increment i
        Add(ptrFreq, sizeUnique, frequency casted as a string)
        Decrement i
```

*Increment i*
*Increment j*

*outputAndSaveConcordance(ptrUnique, ptrFreq, sizeUnique)*
*Print "WORD:", Space, "FREQUENCY", Newline*
*Assign 0 to i*
*Repeat While (i is less than sizeUnique)*
*Print (the value pointed to by ptrUnique[i]), Space, (the value pointed to by ptrFreq[i]), Newline*
*Increment i*
*Print "Where would you like to save the concordance (include .txt)?: ", Newline*
*Read the next line into fileName*
*Repeat While (".txt" isn't present in fileName)*
*Print "Please include .txt: ", Newline*
*Read the next line into fileName*
*Create an output file stream writer*
*Open fileName into writer in the truncate mode*
*If (writer fails to open the file)*
*PrintError "ERROR: Failed to open file . . .", Newline*
*Exit with code -1*
*Print "WORD:", Space, "FREQUENCY:", Newline into writer*
*Assign 0 to i*
*Repeat While (i is less than sizeUnique)*
*Print (the value pointed to by ptrUnique[i]), Space, (the value pointed to by ptrFreq[i]), Newline*
*Increment i*
*Print "Saving performed successfully . . .", Newline*
*Close writer*

*deleteArray(ptr, size)*
*Assign 0 to i*
*Repeat While(i is less than size)*
*Deallocate the memory pointed to by ptr[i]*
*Increment i*
*Deallocate the array pointed to by ptr*

*Main()*
*Assign 0 to size*
*Assign 0 to sizeUnique*
*Allocate memory from the heap for a string pointer array of size size and assign it's the address of*
*its first element to ptrArr*
*Allocate memory from the heap for a string pointer array of size sizeUnique and assign it's the*
*address of its first element to ptrFreq*
*Allocate memory from the heap for a string pointer array of size sizeUnique and assign it's the address of its first element to ptrUnique*
*readFile(ptrArr, size)*

*sort(ptrArr, size)*
*concordance(ptrArr, ptrFreq, ptrUnique, size, sizeUnique)*
*outputAndSaveConcordance(ptrUnique, ptrFreq, sizeUnique)*
*deleteArray(ptrArr, size)*
*deleteArray(ptrFreq, sizeUnique)*
*deleteArray(ptrUnique, sizeUnique)*

## Step 4 – C++ Code:

```
//*************************************************************//
//** Name: Dhiyaa Al Jorf                                    **//
//** Net ID: da2863                                          **//
//** Date Created: November 2, 2021                          **//
//** Assignment 3: Computer Engineering Case Study:          **//
//** Concordance of a text file                              **//
//** Program: Assignment3.cpp                                **//
//*************************************************************//

#include <iostream>
#include <fstream> // To read from and write into files
#include <string> // To manipulate strings
#include <iomanip> // To change the format of the output
#define W 25 // Width for setw();

using namespace std;

// Function Prototypes
void readFile(string**& ptrArr, int& size);
void add(string**& ptrArr, int& size, string value);
void trim(string str, int& size, string**& ptrArr);
void sort(string**& ptrArr, int size);
void swap(string*& x, string*& y);
void concordance(string** ptrArr, string**& ptrFreq, string**& ptrUnique, int size, int&
sizeUnique);
void outputAndSaveConcordance(string** ptrUnique, string** ptrFreq, int sizeUnique);
void deleteArray(string** ptr, int size);

// Main Function
int main() {
	int size = 0, sizeUnique = 0; // Both sizes start at 0
	// Initializing the arrays
	string** ptrArr = new string * [size];
	string** ptrFreq = new string * [sizeUnique];
	string** ptrUnique = new string * [sizeUnique];
	// Reading strings into the array and editting them as required
	readFile(ptrArr, size);
	// Sorting the array of strings
	sort(ptrArr, size);
	// Filling in the ptrUnique and ptrFreq arrays with concordance info
	concordance(ptrArr, ptrFreq, ptrUnique, size, sizeUnique);
	// Saving and outputting the concordance
	outputAndSaveConcordance(ptrUnique, ptrFreq, sizeUnique);
	// Deleting all memory we previously allocated on the heap
	deleteArray(ptrArr, size);
	deleteArray(ptrFreq, sizeUnique);
	deleteArray(ptrUnique, sizeUnique);
	return (0);
}

// readFile() reads all the strings from a user-specified file and performs necessary
edits through the trim() function before saving the strings into an array
void readFile(string**& ptrArr, int& size) {
	// Ask the user for the file to find the concordance of
	string fileName;
```

```
	cout << "Enter the file name (include file extension): ";
	getline(cin, fileName);
	ifstream reader(fileName, ios::in);
	// If the file doesn't exist, output an error message and exit the program
	if (reader.fail()) {
		cerr << "ERROR: Failed to open file . . ." << endl;
		exit(-1);
	}
	string str;
	// Loop to read all strings, perform necessary edits, then save them into an array
	while (reader >> str) {
		trim(str, size, ptrArr);
	}
	reader.close();
}

// add() expands an array and adds a specified value to the end
void add(string**& ptrArr, int& size, string value) {
	string** tempPtr = new string * [size + 1]; // Allocate memory in the stack for an
array with a size 1 more than the arrays current size
	for (int i = 0; i < size; i++) {
		tempPtr[i] = ptrArr[i]; // Copy all the values from the smaller array to
the larger array
	}
	tempPtr[size] = new string(value); // Add a string to the last element of the
array
	delete[] ptrArr; // Delete the smaller array
	ptrArr = tempPtr; // Make the pointer point to the larger array
	size++;
}

// trim() tokenizes the incoming string and saves the tokens into the array
void trim(string str, int& size, string**& ptrArr) {
	int start = 0;
	for (int i = 0; i < str.length(); i++) { // Goes through all the characters of a
string (which can be treated as a character array)
		if (!isalpha(str[i]) || (i == (str.length() - 1) && isalpha(str[i]))) { //
(Checks if the current charcter is non-alphabetical) or (the last character and
alphabetical)
			if (isalpha(str[i])) {
				str[i] = tolower(str[i]);
				i++;
			}
			char* res = new char[(i + 1) - start]; // Character array of a size
1 more than the number of characters in the word to be added (the extra character is for
the \0 character)
			for (int j = 0; j < i - start; j++)
				res[j] = str[j + start];
			res[i - start] = '\0'; // Sets the last character as the end of
string character so it can be saved into a string
			string result = res;
			delete[] res; // Deallocate the memory set for res
			start = i + 1;
			if (result != "") // Checks if the word isn't empty. That can happen
in cases in which the whole string is nonalphabetical
				add(ptrArr, size, result); // Add the cut out portion to the
array
```

```
		}
		else {
			str[i] = tolower(str[i]); // Turns the current letter into it's
lower case form
		}
	}
}

// sort() sorts an array using selection sort
void sort(string**& ptrArr, int size) {
	for (int i = 0; i < size - 1; i++) {
		int minIndex = i;
		for (int j = i + 1; j < size; j++) {
			if ((*ptrArr[minIndex]) > (*ptrArr[j]))
				minIndex = j;
		}
		swap(ptrArr[minIndex], ptrArr[i]);
	}
}

// swap() swaps 2 strings
void swap(string*& x, string*& y) {
	string* temp = x;
	x = y;
	y = temp;
}

// Finds the unique words in an array and the frequency of each unique word
void concordance(string** ptrArr, string**& ptrFreq, string**& ptrUnique, int size, int&
sizeUnique) {
	// Outer loop with 2 counters: i is used for ptrArr and j is used for the
concordance related arrays
	for (int i = 0, j = 0; i < size; i++, j++) {
		add(ptrUnique, sizeUnique, (*ptrArr[i])); // Add a new string to ptrUnique
		sizeUnique--; // The add function increases the size variable but we need
the size to stay the same because it will now represent the size of ptrFreq
		int frequency = 1;
		// Inner loop increments frequency as long as the current string in ptrArr
is the same as the current string in ptrUnique
		for (i++; i < size && *ptrUnique[j] == *ptrArr[i]; i++) {
			frequency++;
		}
		add(ptrFreq, sizeUnique, to_string(frequency)); // Add the frequency
		i--; // Correct for the extra increment that happens when the inner loop
condition becomes false
	}
}

// Outputs the concordance on the output screen and writes it into a file
void outputAndSaveConcordance(string** ptrUnique, string** ptrFreq, int sizeUnique) {
	// Outputing to the output screen in tabular format
	cout << left << setw(W) << "WORD:" << setw(W) << "FREQUENCY:" << endl;
	for (int i = 0; i < sizeUnique; i++) {
		cout << setw(W) << *ptrUnique[i] << setw(W) << *ptrFreq[i] << endl;
	}
	// Ask the user for where to save the file
	string fileName;
```

```
	cout << "Where would you like to save the concordance (include .txt)?: ";
	getline(cin, fileName);
	// The loop makes sure the user is saving the file as a .txt file
	while (fileName.find(".txt") == string::npos) { // The find() function returns
string::npos if the latter string isn't found in the former string
		cout << "Please include .txt: ";
		getline(cin, fileName);
	}
	ofstream writer(fileName, ios::trunc);
	if (writer.fail()) {
		cerr << "ERROR: Failed to create the file . . ." << endl;
		exit(-1);
	}
	// Saves the concordance in tabular form into the user-specified file
	writer << left << setw(W) << "WORD:" << setw(W) << "FREQUENCY:" << endl;
	for (int i = 0; i < sizeUnique; i++) {
		writer << left << setw(W) << *ptrUnique[i] << setw(W) << *ptrFreq[i] <<
endl;
	}
	// Confimation message
	cout << "Saving performed successfully . . ." << endl;
	writer.close();
}

// Deallocates all the memory used by an array of string pointers including the space
used for the strings
void deleteArray(string** ptr, int size) {
	for (int i = 0; i < size; i++) {
		delete ptr[i]; // Deallocates all strings that the pointers in the array
are pointing to
	}
	delete[] ptr; // Deallocates the block of memory that was used for the array of
string pointers
}
```

# Step 5 – Software Testing & Verification:

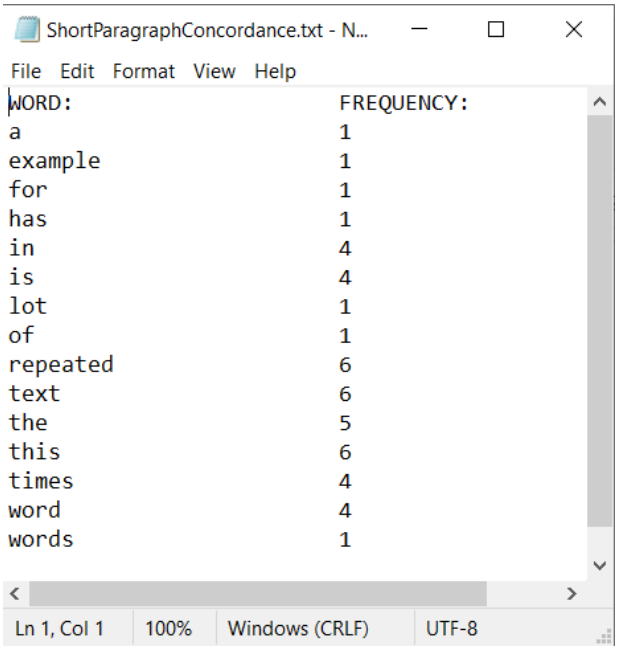
## Case 1 – Input File not present:

```
Microsoft Visual Studio Debug Console
Enter the file name (include file extension): NonexistantFile.txt
ERROR: Failed to open file . . .

D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.3\Assignment3\Debug\Assignment3.exe (process 20552) exit
ed with code -1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

The program threw an error message then ended the execution => This test case passes.

## Case 2 – Short Paragraph:

```
Microsoft Visual Studio Debug Console
Enter the file name (include file extension): ShortParagraphTest.txt
WORD:                   FREQUENCY:
a                       1
example                 1
for                     1
has                     1
in                      4
is                      4
lot                     1
of                      1
repeated                6
text                    6
the                     5
this                    6
times                   4
word                    4
words                   1
Where would you like to save the concordance (include .txt)?: ShortParagraphConcordance.txt
Saving performed successfully . . .

D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.3\Assignment3\Debug\Assignment3.exe (process 3968) exite
d with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```




| WORD: | FREQUENCY: |
|---|---|
| a | 1 |
| example | 1 |
| for | 1 |
| has | 1 |
| in | 4 |
| is | 4 |
| lot | 1 |
| of | 1 |
| repeated | 6 |
| text | 6 |
| the | 5 |
| this | 6 |
| times | 4 |
| word | 4 |
| words | 1 |

The user was prompted to enter the name of the input and output files + The program printed the concordance on the screen + The program saved the concordance into the file + The concordance is sorted alphabetically, contains only lower-case characters, and has the correct frequency => This test case passes.

## Case 3 – Long Block of Text:

Microsoft Visual Studio Debug Console

```
Enter the file name (include file extension): LongParagraphTest.txt
WORD:                   FREQUENCY:
according               1
amplitude               4
and                     12
as                      1
average                 2
avoid                   1
be                      18
before                  1
both                    4
can                     1
configuration           4
constraints             4
data                    1
decimal                 2
different               1
duration                5
file                    2
following               1
for                     1
form                    2
generating              1
give                    2
greater                 5
gust                    3
having                  1
in                      3
input                   1
max                     8
microburst              4
min                     4
minimum                 4
must                    14
negative                6
never                   1
no                      2
non                     5
positive                9
probability             4
sample                  1
should                  6
size                    1
speed                   5
step                    1
storm                   4
storms                  2
than                    5
the                     17
times                   1
to                      3
us                      2
validated               1
value                   1
wind                    2
written                 2
zero                    1
Where would you like to save the concordance (include .txt)?: LongParagraphConcordance
Please include .txt: LongParagraphConcordance.txt
Saving performed successfully . . .

D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.3\Assignment3\Debug\Assignment3.exe (process 26348) exited with code 0.
```

LongParagraphConcordance.txt - Notepad

File Edit Format View Help

| WORD: | FREQUENCY: |
|---|---|
| according | 1 |
| amplitude | 4 |
| and | 12 |
| as | 1 |
| average | 2 |
| avoid | 1 |
| be | 18 |
| before | 1 |
| both | 4 |
| can | 1 |
| configuration | 4 |
| constraints | 4 |
| data | 1 |
| decimal | 2 |
| different | 1 |
| duration | 5 |
| file | 2 |
| following | 1 |
| for | 1 |
| form | 2 |
| generating | 1 |
| give | 2 |
| greater | 5 |
| gust | 3 |
| having | 1 |
| in | 3 |
| input | 1 |
| max | 8 |
| microburst | 4 |
| min | 4 |
| minimum | 4 |
| must | 14 |
| negative | 6 |
| never | 1 |
| no | 2 |
| non | 5 |
| positive | 9 |
| probability | 4 |
| sample | 1 |
| should | 6 |
| size | 1 |
| speed | 5 |
| step | 1 |
| storm | 4 |
| storms | 2 |
| than | 5 |
| the | 17 |
| times | 1 |
| to | 3 |
| us | 2 |
| validated | 1 |
| value | 1 |
| wind | 2 |
| written | 2 |
| zero | 1 |

The user was prompted to enter the name of the input and output files + The program printed the concordance on the screen + The program saved the concordance into the file + The concordance is sorted alphabetically, contains only lower-case characters, and has the correct frequency (when compared to the output of an online concordance generator) => This test case passes.

## Case 4 – Special-Character Rich File

```
Microsoft Visual Studio Debug Console
WORD:                   FREQUENCY:
al                      1
are                     2
be                      1
currently               1
dhiyaa                  1
doing                   2
following               1
hello                   3
how                     2
into                    1
is                      2
jorf                    1
line                    1
m                       1
my                      1
name                    1
p                       1
previous                1
should                  1
split                   1
the                     3
there                   2
time                    1
words                   1
you                     2
Where would you like to save the concordance (include .txt)?: Special Characters Concordance.txt
Saving performed successfully . . .

D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.3\Assignment3\Debug\Assignment3.exe (process 3968) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debuggi
ng stops.
Press any key to close this window . . .
```

Special Characters Concordance.txt - Notepad

File Edit Format View Help

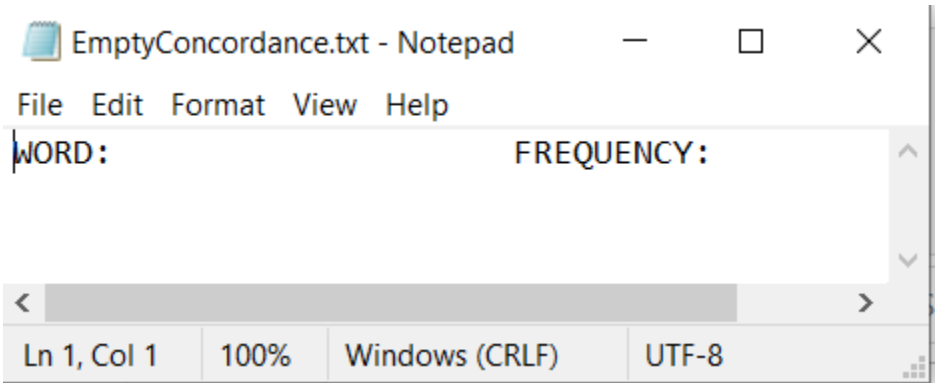
| WORD: | FREQUENCY: |
|---|---|
| al | 1 |
| are | 2 |
| be | 1 |
| currently | 1 |
| dhiyaa | 1 |
| doing | 2 |
| following | 1 |
| hello | 3 |
| how | 2 |
| into | 1 |
| is | 2 |
| jorf | 1 |
| line | 1 |
| m | 1 |
| my | 1 |
| name | 1 |
| p | 1 |
| previous | 1 |
| should | 1 |
| split | 1 |
| the | 3 |
| there | 2 |
| time | 1 |
| words | 1 |
| you | 2 |

The user was prompted to enter the name of the input and output files + The program printed the concordance on the screen + The program saved the concordance into the file + The concordance is sorted alphabetically, contains only lower-case characters, and has the correct frequency (when compared to the output of an online concordance generator) => This test case passes.

**<u>Case 5 – Empty Text File:</u>**

Microsoft Visual Studio Debug Console

```
Enter the file name (include file extension): EmptyTest.txt
WORD:                   FREQUENCY:
Where would you like to save the concordance (include .txt)?: EmptyConcordance.txt
Saving performed successfully . . .

D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.3\Assignment3\Debug\Assignment3.exe (process 18648) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debuggi
ng stops.
Press any key to close this window . . .
```




The output file shows no words are in the string => This test case passes.

## User Guide:

- Make sure to store the input file in the same folder as the Assignment3.cpp file.
- Compile and run the file called Assignment3.cpp
- The program will ask you to enter the name of the input file. Make sure to include the file extension.
- The program will output the concordance of the input file on the output screen.
- The program will then ask for the name of the output file and will save the concordance file accordingly.