

# THE UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE  
LONDON CANADA

## *Software Tools and Systems Programming*

(Computer Science 2211a)

### *ASSIGNMENT 4*

Due date: Thursday, November 12, 2020

11:55 pm Eastern Standard Time – 4:55 am Greenwich Mean Time)

The purpose of this assignment is to provide the student with experience with structures and dynamic memory in a C program. It will incorporate lessons learned on header files and libraries.

**The assignment will be contained within multiple files.**

Performing any word processing capability requires the program to store an indeterminate amount of data for retrieval and editing.

This assignment will provide an exploration of what it takes to manage the input and storage of this type of data.

The amount of data is unknown, and the program must be able to handle any volume of data. This means the program will have to allow for any amount of text.

The program will allow for any number of lines of text, which also means it must handle any number of words per line of text.

Your program must be written using dynamically allocated memory to process anywhere from one to an unlimited number of lines of text. Each line must be able to process anywhere from one to an unlimited number of words. Therefore, using a set size array of characters or a set size array of lines is not viable. No constant values can be used in the array declarations.

The mechanism we will use to store these words and sentences will be based on specific structures. The structures must be defined using typedef in order to create variable aliases for each structure. Each structure can ONLY contain the three elements listed in each – do NOT add any other elements to either structure.

The structure for the `word` construct is:

- 1.) a pointer to a character
- 2.) an integer to store the number of characters in the word
- 3.) an integer to store the position of the word in the sentence it is contained in.

The structure for the `sentence` construct is:

- 1.) a pointer to the `word` construct listed above
- 2.) an integer to store the which line this is in the total collection
- 3.) an integer to store the exact number of words in this line

(Yes, I am aware most of these can be computed by the program. But we need something for the structures to contain and these are all relevant to the program task.)

**Your program must:**

- prompt the user for a line of text.
- process each line one by one by:
  - breaking up each word in each line of text (ignoring ALL white space)
  - store each word separately in an instance of the word structure
  - for each word, store
    - the size (number of characters) in that word
    - the position in that line of this specific word
- repeat this process until the user enters an empty line (hits the enter key without text).

After the user enters an empty prompt (hits the enter key on a blank line), the program must:

- print each and every word out, each word on a single line, in the order entered by the user.
- print out each sentence in its entirety in the order it was entered by the user.

Next, the program will prompt the user for a word to search for in the text entered.

- if it finds the word, it will print out which line and position the word was found in.
  - it must search the entire text and report each time the word is found.
  - it must use the values stored in the structures.
- it must inform the user if it does not find the word.
- repeat this process until the user enters an empty line (hits the enter key without text).

Because this is of unknown size, the program can ONLY use dynamic memory for the entire program. The program can NOT have ANY array declarations ( i.e. `array[10];` ). Every part of the program MUST be dynamically allocated.

[ **NOTE:** There is ONE (1) and only one exception. To keep this assignment easy, the user can declare the input character array of a specific set size. for example, assuming the input string array is labeled `str` then the line:

```
char str[1000];
```

is allowed.

**BUT !** This is the only exception. Even the members of each structure listed above will be pointers and NOT set arrays.

The algorithm is:

- for each new line to be processed a portion of memory will be dynamically allocated for the `sentence` structure above.

- as each word is processed, a portion of memory will be dynamically allocated for the word structure above.
- as each word is processed, a portion of memory will be dynamically allocated for the character array that is in the word structure above.

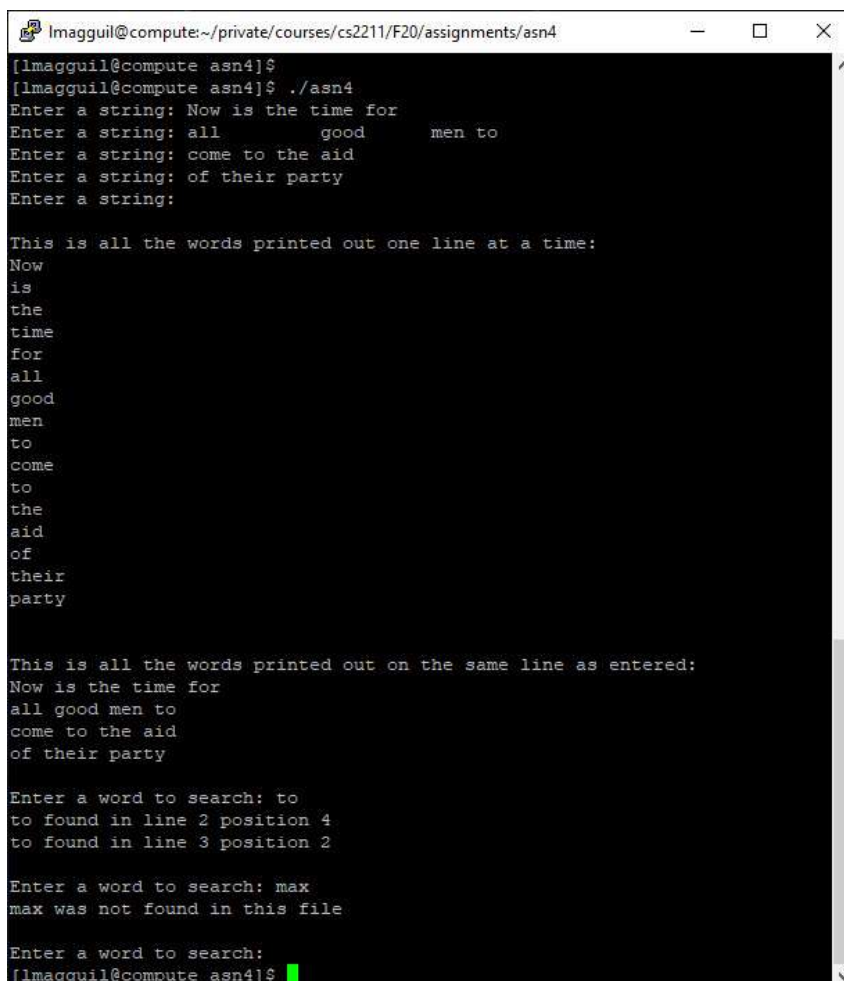
You are **NOT** allowed to use linked list in this assignment. Everything is to be memory that is dynamically allocated at run time.

To accomplish this, you will need to use the `realloc()` function in C.

[NOTE: keep in mind how `realloc()` works. The added amount of dynamic memory allocated in the heap **MUST** include the memory already allocated for that pointer **PLUS** the new memory. So, if `*myPointer` already had 64 bytes allocated and you needed to expand that block by 32 more bytes, then the requested memory size would be 96 ( $64 + 32$ ) and NOT 32.

```
myPointer =  
(variableType *) realloc (myPointer, (96 * sizeof(variableType)));
```

**Here is an example of the program:**



```
lmagguil@compute:~/private/courses/cs2211/F20/assignments/asn4  
[lmagguil@compute asn4]$  
[lmagguil@compute asn4]$ ./asn4  
Enter a string: Now is the time for  
Enter a string: all      good      men to  
Enter a string: come to the aid  
Enter a string: of their party  
Enter a string:  
  
This is all the words printed out one line at a time:  
Now  
is  
the  
time  
for  
all  
good  
men  
to  
come  
to  
the  
aid  
of  
their  
party  
  
This is all the words printed out on the same line as entered:  
Now is the time for  
all good men to  
come to the aid  
of their party  
  
Enter a word to search: to  
to found in line 2 position 4  
to found in line 3 position 2  
  
Enter a word to search: max  
max was not found in this file  
  
Enter a word to search:  
[lmagguil@compute asn4]$
```

Your prompts can be as simple as this, or more descriptive. This is totally optional and up to your discretion. The above shows the minimum allowed prompts.

Notice in line 2 of the input above there are multiple spaces between words. ALL spaces are to be omitted. Only words are allowed.

Punctuation is to be ignored (if possible, do NOT use punctuation in your input. This would have added too much complexity to the assignment.) Assume no punctuation will be inputted.

This program must allow for an unlimited number of lines and an unlimited number of words per line. BUT, keep the testing within reason.

**Your program must be** broken down into logical units as per the class notes and previous lab/assignment.

The main.c should be a ‘control’ program that does little or no processing and calls on functions to perform the programmed tasks.

You are NOT allowed to use global variables in your program.

hint: there should be a single `#include` in the main.c

The files in my solution were:

- headers.h
- definitons.h
- utilities.c
- inputFunctions.c
- outputFunctions.c
- main.c

hint: these file names are only (merely suggestions). You do NOT have to use any or all of these file names (except main.c of course) You can break down the code in any manner you see fit, as long as they are not all in a single main.c

**Remember:** This is all done with dynamic memory.

Except for the one input array for the user’s text, you will lose major marks if an array definition (i.e. brackets in the array definitions that have constants – example `word[30];` )

There one exception as mentioned above, for the input character string array for the input of the text string (and remember, you can re-use this character array for the input for the search word also...)

## Working in UNIX.

1. Type the following to begin recording your session in a file called `yourUserName_Asn4.script`

**script YourUserName\_Asn4.script**

- (using your actual user name).

2. Display the current date and time using the appropriate command

3. Display your username using the appropriate command

4. Display the contents of the current working directory

5. Compile the program again

6. Run the program with some meaningful examples.

7. Type **exit** to stop your screen capture session.

**note:** do NOT rename the main.c file (keep the file name main.c – no action is required on your part.)

## Required Coding Standards

All code is to be indented correctly.

Comments at the very beginning (top – first lines) of the file must be:

```
/* CS2211a 2020 */
/* Assignment 04 */
/* your name */
/* your student number */
/* your UWO Account Name */
/* Date Completed */
```

All variables **MUST** have a comment describing their intended use(s).

A comment describing the code for each function must be included.

The comment(s) can be brief but must convey what that section of code performs.

## Submission Instructions:

You must upload and submit, via the CS2211A OWL Web Site, any files required to compile your code along with the script file you created:

Change your current working directory to `~/courses/cs2211a/Asn`. Create a file named `YourUserName_asn4.tar.gz` and submit this file for assignment 4. YourUserName should be your UWO email account user name which is the same as your Gual account login user name. (For detailed information, please check CS2211a Assignment Submission Guidelines).

It is the student's responsibility to ensure the work was submitted and posted in OWL. OWL replies with a summation verification email (every time).

Any assignment **not** submitted correctly will **not** be graded.

The teaching assistant grading your assignment will compile and run your program.

**If the program does not compile, the TA will NOT attempt to correct or fix your program so it will run.**

**AND AGAIN PLEASE: Do not cheat or copy.**

Remember, these are NOT community projects but are expected to be completed individually.