WESTERN UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE
LONDON CANADA

Software Tools and Systems Programming

(Computer Science 2211a)

ASSIGNMENT 5

Due date: Monday, November 30, 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.

(Basically, Assignment Five is simply take your assignment Four code, and instead of using realloc() to manage any size of input, you will use linked lists. The requirements in red below show the differences in assignment five.) Crossed out lines are there to display the difference.

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~~ must at least contain the four elements listed in each ~~do NOT add any other elements to either structure~~.

The structure for the `word` **node** construct is:

1.) a pointer to a character string
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.
4.) a pointer to the next word in the sentence.

The structure for the `sentence` **node** construct is:

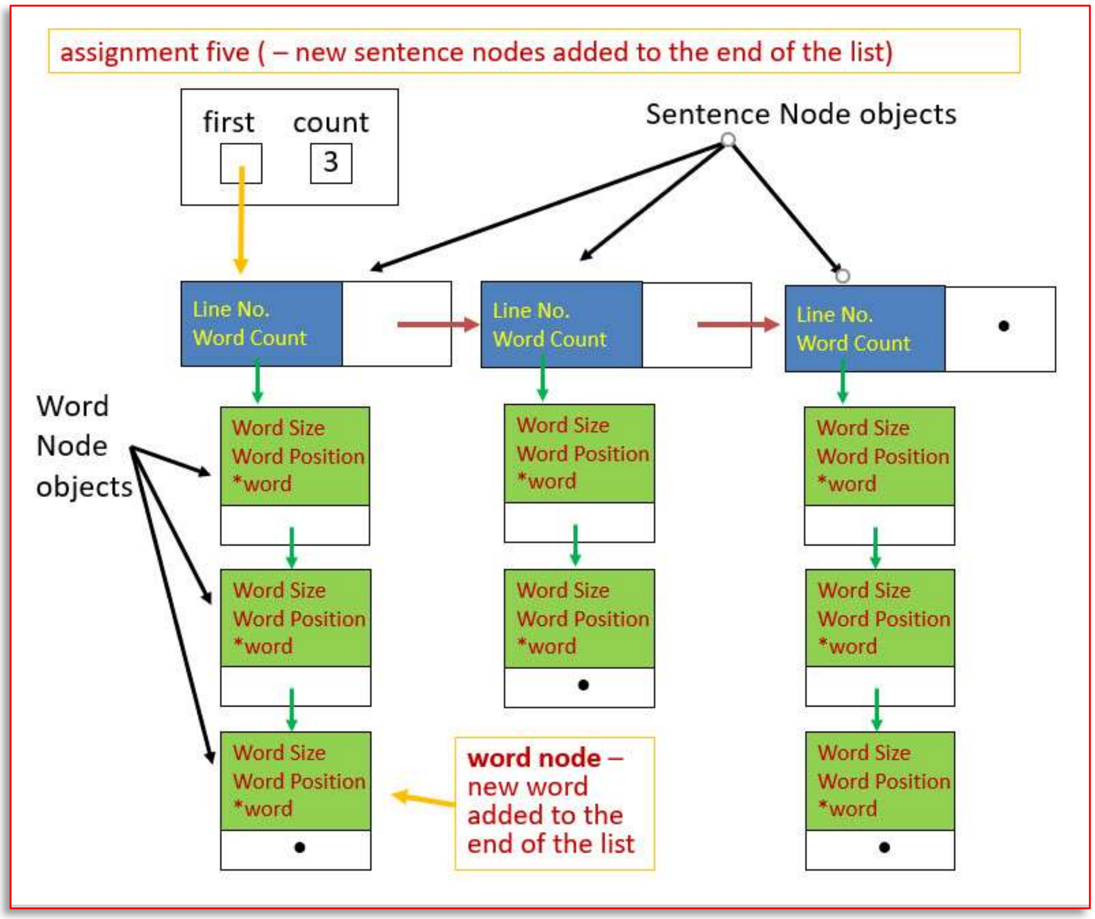
1.) a pointer to the `word` construct listed above
2.) an integer to store the which line this is in the total collection
3.) an integer to store the exact number of words in this line
4.) a pointer to the next sentence in the list.

(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.)

The control structure (see class notes) for the `linked list` is:

1.) a pointer to the FIRST sentence entered
3.) an integer to store the exact number of current sentences in this linked list.

This is a representation of the final construct for these linked lists:


assignment five (– new sentence nodes added to the end of the list)
first
count
3
Sentence Node objects
Line No.
Word Count
Word Node objects
Word Size
Word Position
*word
word node – new word added to the end of the list

Your program must:

- prompt the user for a line of text.
- process each line one by one by:
    - create the sentence node in dynamic memory and add it to the list of sentences.
    - breaking up each word in each line of text (ignoring ALL white space)
    - store each word separately in a node of the word structure
        - for each word,
            - create the word node in dynamic memory and add to list of words.
            - then store the size (number of characters) in that word
                - the position in that line of this specific word
                - the character string in the word pointer in this structure.
- 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.~~

(do NOT print out each word separately for this assignment!)

- 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).

Next, prompt the user for the number of a line to delete (i.e. 3 for the third line, 1 for the first). Print out each of the remaining sentences in their entirety in the order it was entered by the user. If the number is greater than the number of sentences, inform the user the request was denied and prompt again.

- repeat this process until the user enters an empty line (hits the enter key without a number).

So – if the current linked list consisted of 5 lines (lines 1, 2, 3, 4 and 5) and the user enters the number 4 to request the removal the fourth line, then the linked list would consist of four lines (lines 1, 2, 3 and 5). You MUST change the line number value in all the remaining lines. In this example, you MUST change the 5 to a 4 in the last node. So now the four remaining lines are numbered 1, 2, 3 and 4. You would then print out lines 1, 2, 3 and 4 to demonstrate the requested line was removed.

– if the current linked list consisted of 6 lines (lines 1, 2, 3, 4, 5 and 6) and the user enters the number 2 to request the removal the second existing line, then the linked list would consist of five lines (lines 1, 3, 4, 5 and 6). You MUST change the line number value in all the remaining lines. In this example, you MUST change the 3 to a 2 in the second existing node, you MUST change the 4 to a 3 in the third existing node, you MUST change the 5 to a 4 in the fourth existing node, you MUST change the 6 to a 5 in the fifth existing node So now the five remaining lines are numbered 1, 2, 3, 4 and 5. You would then print out lines 1, 2, 3, 4 and 5 to demonstrate the requested line was removed.

You should **not** need to be instructed to do this last part. Delete all nodes and free all memory before ending the program. Also, comment your code (both of these should be standard by now – please carry these practices with you into your other courses !)

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 using the linked list as demonstrated in the notes. So, retain your code that dynamically allocate the word string using calloc() that was in your assignment four. The actual word (that will be stored in the word node) is still a dynamically allocated string.
[**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.]~~ Does not apply to assignment five.

The algorithm is:
- for each new line to be processed a new sentence node will be dynamically allocated for the `sentence` node structure above.
- as each word is processed, a new word node will be dynamically allocated for the `word` structure node 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~~ MUST use linked list in this assignment. Everything is to be memory that is dynamically allocated at run time, including control structure (see the notes for details).

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

[NOTE: do NOT use realloc() anywhere in this assignment !]

Your prompts can be the same as you used in assignment five. This is totally optional and up to your discretion. The above shows the minimum allowed prompts.

Again, your program must deal with 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.

You are **not** to use break or continue in non-standard ways (i.e. to break out of an intentional infinite loop
like

```
    while (1) {<stuff;>  if (x == 5) break; <other stuff>};
```

This is bad coding and leads to unreadable and 'spaghetti' code.

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 programed 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
- createNode.c
- inputFunctions.c
- outputFunctions.c
- searchWords.c
- removeLine.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

Creating a Makefile to compile the source code

You are asked to create a Makefile to compile your source code. When "make" is typed, an executable program called "myParagraph" is generated. Typing "make clean" delete all the files generated by "gcc".

Working in UNIX.

1. Type the following to begin recording your session in a file called yourUserNAme_Asn5**`.script`**

   script YourUserName_Asn5.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 **using your makefile**

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 05 */
/* 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 or major action 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_asn5.tar.gz and submit this file for assignment 5. 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.