CIS*2500 (Intermediate Programming) Lab #2 – Dynamic memory allocation

Due date: End of Week 5, Friday Feb 12th at 11:59PM EST (Submitted to the GitLab repository)

Description
Your program will allow the user to type in several lines of a poem. The user will signal the end of their input by typing a single dot (".") on a new line. After the end of the input stage, the program will tell the user (on *stdout*) how many lines and words are in the poem and how many words are on each line of the poem. The first line of output provides the total words and lines in the poem, and the second line is a space-delimited list of the number of words on each line of the poem provided. See **Sample Input and Output** for more details.

Sample Input and Output (Outputs in black, inputs in red)

```
$ ./poetry
Enter the poem:
Now is the winter of our discontent
Made glorious summer by this
Sun of York
.
15 words on 3 lines
7 5 3
```

Sample Input and Output Explanation

- The user is prompted to enter the poem into stdout, with each line of the poem on its own line. The user inputs three lines in this example, and the inputs "." and hits enter to end the program (This is the red text above).
- The program them indicates the total number of words in the poem, and the total number of lines inputted (15 and 3, respectively). This is the first of two lines of output.
- The second line is a space-delimited list of the number of words on each line. In this case, the three lines of the poem has seven words on the first line, five words on the second line, and three words on the third line.
- The program ends the space-delimited list with a newline.

Constraints
Input: Lines of poetry

- Contain no punctuation
- No word can be greater than 20 characters in length
- No line can be greater than 100 characters in length
- You must not use static allocation of arrays – they must be allocated dynamically. The number of input lines is unknown, so you must dynamically reallocate your array as the program runs.

Output

- The first line of output will have the following format

  x **words on** y **lines**

  where x is the number of words and y is the number of lines and there are single spaces (blank characters) between each number or word.
  If **words** or **lines** = 1 then **words** should be changed to **word** and **lines** to **line**, respectively.
- The second line of output will have a format as shown in the above sample input / output scenario (basically number of words on each line separated by spaces).

Program Structure

- The program is to be named *poetry.c.* If you choose to use multiple files, then write your main function in a file called *main.c*, all your function prototypes and such in a header file called *header.h*, and all function definitions in a file called *poetry.c*
- You must use a makefile to compile the code and to produce an executable called *poetry*.
- There will be no use of global variables.
- No debugging prints may appear during the execution of *poetry* when it is being graded.

Other Rules

- Your code must compile cleanly with no error or warning messages using -Wall and -std=c99 flags in gcc on the SOCS server
- You **must** use *realloc* when resizing the array that holds the number of words per line. A small deduction will be applied if *realloc* is not used.
- All work is to be done independently. If the TA deems that you have not done the coding independently then they will not grade the assignment.
- Code that cannot be compiled with **your** makefile will not be graded. Your makefile should compile your code when the user inputs "make" into the command line.

Submission Instructions

- Ensure the most recent copy of your program is updated to your GitLab repository, with the following restrictions:
  - If you use a single file, then you must submit *makefile* and your program file called *poetry.c*
  - If you use multiple files, then you must submit *makefile* and your program files (*main.c, header.h, poetry.c)*
  - Your makefile should expect the main.c, header.h and poetry.c files to be located in the same directory as your makefile.
- Recap on submission to Gitlab

  Step 1: Login to the school server and cd to the directory that you work on from the terminal application (or use the method that used to submit lab1).

Step 2: From the chosen directory, now type:

```
git clone https://git.socs.uoguelph.ca/2500w21/<your username>/lab2.git
```

At this point, you have a directory to work with on your account (or on your local system)

Step 3: Do the work

- (remember to) cd to the new directory
- create the file that you are working on
- after first save, type: `git add filename`

ONLY ADD THE FILE ONCE!!! //do not do: git add .

Loop every 20-30 minutes:

```
   git commit -am "write something here about what you just did"
```

Once per day:

```
          git push   // this is what stores local work back to the server.
```

To learn more about Gitlab, go to this link on moodle
https://moodle.socs.uoguelph.ca/course/view.php?id=169

Make and Makefiles

Your makefile should look similar to the following (if you use a single file called poetry.c):

```
CC=gcc
CFLAGS=-Wall -std=c99

poetry: poetry.o
	$(CC) poetry.o -o poetry

poetry.o : poetry.c
	$(CC) $(CFLAGS) -c poetry.c

clean :
	rm poetry poetry.o
```

Remember that there is a <TAB> before lines starting with $(CC) and rm. Make may not work if there are spaces and not tabs.

For more on makefile, read the tutorial found here:
https://www.gnu.org/software/make/manual/html_node/Introduction.html#Introduction