CIS*2500 (Intermediate Programming) Lab #2 part A

Due date: in lab (Week 5 – starting Feb 6th)

You must complete part A of the lab outside lab hours before you attend your lab. Part B of the lab will be assigned to you by your lab TA in the beginning of the lab. You must complete part B in the lab, and request for a grade before you leave the lab.

Concepts: Dynamic Memory Allocation, makefile, gitlab

Description – part A

Write a program that takes as input, a text file that has x lines of a poem. The poem ends with a period. After the end of the input stage, the program will print 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.

Two sample input files named poem.txt and poem2.txt are given for convenience.

Sample Input and Output 1

$ ./poetryL2 poem.txt
```
15 words on 3 lines
7 5 3
```

Sample Input and Output 2

$ ./poetryL2 poem2.txt
```
37 words on 6 lines
7 6 6 5 9 4
```

Sample Input and Output Details

- The program is run with the name of the input file as a command-line argument
- The program then indicates the total number of words in the poem, and the total number of lines inputted. This is the first of two lines of 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 is a space-delimited list of the number of words on each line, as shown in the above sample input / output scenario.
- The program ends the space-delimited list with a newline.

Constraints and assumptions

Input file: Lines of poetry

- Contain no punctuation (except the period to indicate the end of the poem).
- 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.

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 *poetryL2*.
- DO NOT use 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.
- 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.

Note: Refer to tutorials and lectures on makefile and gitlab submissions.