The task is to implement a word-counting tool. It should parse an input file into individual words, count the number of occurrences of each word, and then print n most frequent words with a minimal length of **m** characters, along with their counts.
The name of the input file will be provided as the first command-line argument. The second command line argument is the number of the most frequent words to be included in the output. The third command line argument will specify the minimal length of a word to be considered. The provided file will contain words separated by spaces, so one can expect words composed of alphabetical characters only, separated by a single space. For the sake of simplicity, assume that the words are at most 50 characters long. To tune your program, you can use the **example_input_short.txt** and **example_input_long.txt** files provided in the assignment files. The assignment files also contain examples of the output as **example_output_[type]_[n]_[m].txt**.

You have to zip and upload your solution to the brute system.

Suggested, but optional walkthrough:

1) Download and unzip the assignment files (e.g. by "**`wget https://cw.fel.cvut.cz/wiki/_media/courses/be5b99cpl/lectures/test_cpl_2025-11-19.zip`**"). Check their structure and create a compilable `main.c`, which accepts three command line arguments or produces an error message to standard error output. **(1 point)**

2) Allocate sufficient memory and read in the input file (hint: fopen, realloc). **(1 points)**

3) Create a suitable data structure to contain the words and their counts (hint: typedef struct) **(1 point)**

4) Parse the contents of the input file to obtain individual words and count them (hint: use the fact that C string is terminated by a special character) **(2 points)**

5) Print out **n** (provided as a second command-line argument) most used words with length at least **m** (provided as a third command-line argument) along with their count **(3 points)** (hint: man qsort, also see point 8).

6) Place all the functions in a separate module with a header file and use the main.c only for the main function. **(2 point)**

7) Write a Makefile that compiles and links the module and the main program, and functions have comments regarding their arguments and what the function does. Correct any compiler warnings when compiling with "`-Wall -std=c99 -pedantic`" flags. **(2 points)**

8) Use qsort to sort out the words by their frequency from the most to the least used before printing them **(3 points)**