The task is to implement a simple calculator working in natural language. The calculator should be able to read simple formulas with a single operator and two operands at its standard input and print the results in Arabic numbers onto standard output. For example, an input of "`three thousand twenty five plus forty one`" should result in "`3066`" or, e.g., "`twenty five minus fifteen`" should result in "`10`". The operands will be natural numbers between 1 and 999999, and there will be only two operators in the formulas, **"`plus`"** or **"`minus`".** To characterise the natural language, you will be provided with text files called "**`numbers_[en,wr].txt`**" and "**`orders_[en,wr].txt`**", with each line containing a string and a number that it represents. Moreover, you will be provided with "**`examples_[en,wr].txt`**" with formulas at each line and "**`results_[en,wr].txt`**" with the expected results, again arranged line by line. Your program should accept two command line arguments indicating the files with the language.
You have to zip and upload your solution to the brute system.

Suggested walkthrough:
1) Download and unzip the assignment files (e.g. by "**`wget https://cw.fel.cvut.cz/wiki/_media/courses/be5b99cpl/lectures/test_cpl_2023.zip`**"). Look at their structure and create a compilable `main.c`, which accepts two command line arguments; otherwise, it produces an error message to standard error output. **(1 point)**

2) Create a suitable data structure to represent the numbers and read in the contents of the "`numbers_[en,wr].txt`" and "`orders_[en,wr].txt`" files. Check if your program can translate, e.g., "`one`" or "`ten`" (or "usa" or "trece"`)` to the relevant numbers. **(2 points)**

3) Make your code capable of processing inputs consisting of several words, "`twenty two thousand three hundred twenty five`" and check if it can convert them into numbers. **(2 points)**

4) Add the capability to process an operator, which is either "`plus`" or "`minus`". This operator is independent of the language. Check for both single- and multiple-word operands. **(1 point)**

5) Process all the lines in the "`examples_en.txt`" and check if it produced the same result as the contents of "`results_en.txt`". Do the same for "`examples_wr.txt`" and "`results_wr.txt`". Indicate if there are any errors in the provided examples. **(2 points)**

6) Place the functions from point 3 into another module with a header file. Use include guards for safety. **(1 point)**

7) Write a Makefile that compiles and links the module and the main program, and make sure memory operations are safe (valgrind), and functions have comments regarding their arguments and what the function does. Correct any compiler warnings when compiling with "`-Wall -std=c99 -pedantic`" flags. **(1 point)**

Extras:.

8) Add an option to output the result in a natural language as specified in the files. **(+ 2 points)**
9) Capability to process the complete input “`examples_[en,wr].txt`” files and not only individual lines. **(+ 1 points)**