Your task is to decipher a matrix of text that is encrypted by shuffling its columns. You are provided with three input files '**example.txt**', '**input-0.txt**' and '**input-1.txt**' containing the encrypted text arranged in lines of the same length. Moreover, you are provided with files '**example.perm**' and '**input.perm**' indicating the possible permutations of the columns. Furthermore, you are provided with files '**example.lang**' and '**input.lang**', containing example text in the original language of the encrypted text.

Your task is to write a program, capable of deciphering the input files, i.e., to rearrange their columns according to the provided permutations and then use the language example to verify which of the permutations are valid. Your program should have three command line arguments indicating the processed files and it should print the deciphered text as one sentence per line.
You have to upload your solution to the brute system.
New line characters are not considered as part of the encrypted text.

Suggested walkthrough - each step, if implemented correctly, grants you one point. An alternative walkthrough is doing points 1-5 first, then 9 and 10 and finally 6, 7 and 8.

1) Examine the example functionality on the next page, download and unzip the assignment files (e.g. by "wget https://cw.fel.cvut.cz/wiki/_media/courses/be5b99cpl/lectures/test_2022.zip"). Password is "G4ngn4mStyl3!". Have a look at their structure and create a compilable **main.c**, which accepts three command line arguments, otherwise produces an error message.
2) Read in the permutation file containing the permutation indices until the first occurrence of '---' which signifies the end of a sequence. We suggest keeping the file open to speed up reading the subsequent permutations in points 6-8.
3) Read in the encrypted file, apply the first permutation of columns on each line, save it in an array and print the resulting output.
4) Create a function that will print the deciphered contents in a nicer way: One sentence per line without multiple spaces between words.
5) Decipher the 'input-0.txt' using the first permutation in 'input.perm'.
6) Create a function that will search for the deciphered words in the vocabulary file 'input.lang' and count the score of a particular permutation as a number of words found in the vocabulary.

7) Make sure you are searching for complete words, not their fragments, e.g. when searching for ’or’ do not count a partial match with ‘world’.
8) Decipher the 'input-1.txt' file using the permutations in ‘input.perm' and vocabulary in 'input.lang'. Be cautious about reading from the files as repeated file opening and seeking content might take a bit more time, so think about better options.
9) Place the functions from points 4 and 6 in another module with a header file.
10) Write a Makefile that compiles and links the module and the main program, and make sure memory operations are safe (check the output of valgrind), and functions have comments regarding their arguments and what the function does. Correct any default compiler warnings.

Example functionality:

An example is provided in the 'example.txt', 'example.col' and 'example.lang'

Contents of 'example.txt':

`ASCII text`	`Hex values`

`cab`
`iint`
	`0x63 0x61 0x62 0x20 0x0a`
	`0x69 0x69 0x6e 0x74 0x0a`

Contents of 'example.perm’:

`ASCII text`	`Hex values`

`1`
`2`
`0`
`3`
`---`
`0`
`3`
`1`
`2`
`---`
	`0x31 0x0a`
	`0x32 0x0a`
	`0x30 0x0a`
	`0x33 0x0a`
	`0x2d 0x2d 0x2d 0x0a`
	`0x30 0x0a`
	`0x33 0x0a`
	`0x31 0x0a`
	`0x32 0x0a`
	`0x2d 0x2d 0x2d 0x0a`

Contents of 'example.lang'

ASCII text	Hex values
init	**0x20 0x69 0x6e 0x69 0x74 0x20 0x0a**

Applying the two permutations from ‘example.col’ would result in two possible rearrangements:

ASCII text	Hex values
abc **init**	**0x61 0b62 0x63 0x20 0x0a** **0x69 0x6e 0x69 0x74 0x0a**
c ab **itin**	**0x63 0x20 0x61 0x62 0x0a** **0x69 0x74 0x69 0x6e 0x0a**

where the first rearrangement would be more likely a correct one as it contains the word 'init', which is in the example language.
Therefore, the output of your program should be:

ASCII text	Hex values
abc init	**0x61 0b62 0x63 0x20 0x69 0x6e 0x69 0x74 0x0a**