Chapter V

Mandatory part

- Write a function that returns a line read from a file descriptor.
- What we call a “line” is a succession of characters that end with ’\n’ (ascii code 0x0a) or with End Of File (EOF).

https://latedev.wordpress.com/2012/12/04/all-about-eof/ : taht article can prove useful for someone who can read.

- Your function must be prototyped as follow :

```
int     get_next_line(const int fd, char **line);
```

- The first parameter is the file descriptor that will be used to read.
- The second parameter is the address of a pointer to a character that will be used to save the line read from the file descriptor.
- The return value can be 1, 0 or -1 depending on whether a line has been read, when the reading has been completed, or if an error has happened respectively.
- Your function get_next_line must return its result without ’\n’.
- Calling your function get_next_line in a loop will then allow you to read the text available on a file descriptor one line at a time until the end of the text, no matter the size of either the text or one of its lines.
- Make sure that your function behaves well when it reads from a file, from the standard output, from a redirection etc.
- In you header file get_next_line.h you must have at least the prototype of the function get_next_line and a macro that allows to choose the size of the reading

buffer for the `read` function. This value will be modified during the defence to evaluate the strength of your function. That macro `must` be named BUFF_SIZE. For example:

```
#define BUFF_SIZE 32
```

Is your function still working if your BUFF_SIZE value is 9999 ? And if BUFF_SIZE is valued 1 ? And 10000000 ? Do you know why ?

- We consider that `get_next_line` has an undefined behavior if, between two calls, the same file descriptor designs two distinct files although the reading from the first file was not completed.
- We consider also that a call to `lseek(2)` will never take place between two calls of the function `get_next_line` on the same file descriptor.
- Finally we consider that `get_next_line` has an undefined behavior when reading from a binary file. However, if you wish, you can make this behavior coherent.
- Global variables are forbidden.
- Static variables are allowed.

To know what is a static variable is a good start: https://en.wikipedia.org/wiki/Static_variable

Chapter VI

Bonus part

The project `get_next_line` is straight forward and leaves very little room for bonuses, but I am sure that you have a lot of imagination. If you ace perfectly the mandatory part, then by all means complete this bonus part to go furher. I repeat, no bonus will be taken into consideration if the mandatory part isn't perfect.

- To succeed `get_next_line` with a single static variable.
- To be able to manage multiple file descriptor with your `get_next_line`. For example, if the file descriptors 3, 4 and 5 are accessible for reading, then you can call `get_next_line` once on 3, once on 4, once again on 3 then once on 5 etc. without losing the reading thread on each of the descriptors.