Midterm assignment

Submit Assignment

Due Nov 30 by 11:59pm **Points** 10 **Submitting** a file upload **File Types** zip
Available Nov 17 at 12am - Nov 30 at 11:59pm 14 days

Implementation of command *reverse*

In this task we implement the command *reverse*: the contents of files in command-line arguments, or lines read from the standard output in reverse order with numbering and the lines mirrored. For
example if the content of *test.txt* is this:

```
apple
peach
plum
```

then the result of command *reverse* is:

```
3 mulp
2 hcaep
1 elppa
```

Example for multiple input files: if *test.txt* is given twice as command-line argument then the output is this:

```
3 mulp
2 hcaep
1 elppa
3 mulp
2 hcaep
1 elppa
```

For the solution you may assume that an entire file fits in memory.

The solution can be uploaded arbitrary times until the deadline. The solutions will be compiled by an automated tool and run for different test cases. The results of tests will be seen in Canvas as comments.

Important: the outputs given in this task should match exactly otherwise the automated test tool will evaluate it as a wrong solution!

Task

The program should read the files given as command-line arguments. If the current argument results a fault (i.e. file can't be opened), then print an error message to the standard error output and continue the process with the
next argument. The error message should be `File opening unsuccessful!`.

If the user didn't provide any arguments, then read the lines from the standard input. In this case don't print anything to the console yet, but wait for user input.

(Hint 1: File pointer is technically a stream that can be substituted by standard input (`stdin`).)

(Hint 2: EOF character can be entered with Ctrl+D on Unix systems and Ctrl+Z on Windows.)

Use dynamic array for storing lines! At array creation the array should be a fixed value (e.g. 8). If memory allocation is unsuccessful, then print an error message (`Memory allocation failed!`) and finish execution with some error code.
Don't count the number of lines in advance, but double the array size when the number of lines exceeds the current size. You may assume that the length of a line is not longer than 1024 characters.

Separate the solution to multiple translation units. `main()` function should go to `main.c` and every other function implementation should go to a separate translation unit which includes a header file. Make sure to use header guards.

Requirements of the program

- Non-compiling code gets 0 points automatically. (Of course, this applies to the last solution uploaded, after the task deadline.)
- Don't use global variables!
- Divide the solution logically. Implement the parts of the solution in separate functions.
- Make sure to avoid memory leak!
- Make sure to avoid undefined behaviors!

Suggestions

- Don't forget to check the success of dynamic memory allocation and its deallocation. Test your program with `valgrind` in order to detect potential memory leaks.
- If you use `fgets()` for reading, then don't forget that this function also reads the newline character which may result wrong output when printing.