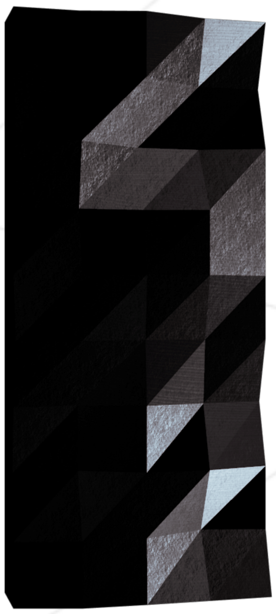

Get Next Line

Reading a line from a fd is way too tedious

fdから行を読むのは面倒すぎる。

Summary:
This project is about programming a function that returns a line read from a file descriptor.

Version: 12

Contents

Chapter I

Goals

This project will not only allow you to add a very convenient function to your collection, but it will also make you learn a highly interesting new concept in C programming: static variables.

このプロジェクトは、あなたのコレクションにとても便利な関数を追加できるようにするだけでなく、C言語プログラミングの非常に興味深い新しい概念、静的関数を学ばせてくれる。

Chapter II

Common Instructions

- Your project must be written in C.
- Your project must be written in accordance with the Norm. If you have bonus files/functions, they are included in the norm check and you will receive a `0` if there is a norm error inside.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.
- If the subject requires it, you must submit a `Makefile` which will compile your source files to the required output with the flags `-Wall`, `-Wextra` and `-Werror`, use cc, and your Makefile must not relink.
- Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To turn in bonuses to your project, you must include a rule `bonus` to your Makefile, which will add all the various headers, libraries or functions that are forbidden on the main part of the project. Bonuses must be in a different file `_bonus.{c/h}` if the subject does not specify anything else. Mandatory and bonus part evaluation is done separately.
- If your project allows you to use your `libft`, you must copy its sources and its associated `Makefile` in a `libft` folder with its associated Makefile. Your project's `Makefile` must compile the library by using its `Makefile`, then compile the project.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done

after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Chapter III

Mandatory part

Function name	`get_next_line`
Prototype	`char *get_next_line(int fd);`
Turn in files	`get_next_line.c, get_next_line_utils.c,` `get_next_line.h`
Parameters	`fd: The file descriptor to read from` 読み込むファイルディスクリプタ
Return value	`Read line: correct behavior` 正しい行動 `NULL: there is nothing else to read, or an error occurred` 他に読むものがないか、エラーが発生しました。
External functs.	`read, malloc, free`
Description	`Write a function that returns a line read from a file descriptor` ファイルディスクリプタを返す関数を書く

get_next_line()関数を繰り返し呼び出す（ループを使うなど）ことで、ファイルディスクリプタが指すテキストファイルを1行ずつ読めるようになるはずだ。

- Repeated calls (e.g., using a loop) to your `get_next_line()` function should let you read the text file pointed to by the file descriptor, **one line at a time**.

関数は、読み込まれた行を返すべき。 他に読み込む行がない場合やエラーが発生した場合は、NULLを返すべき。

- Your function should return the line that was read.
  If there is nothing else to read or if an error occurred, it should return `NULL`.

ファイルを読み込むときも、標準入力から読み込むときも、関数が期待通りに動作することを確認する。

- Make sure that your function works as expected both when reading a file and when reading from the standard input.

ファイルの末尾に到達、且つ「\n」文字で終わらないとき以外、return行は末尾の「\n」文字を含むことに注意。

- **Please note** that the returned line should include the terminating `\n` character, except if the end of file was reached and does not end with a `\n` character.

get_next_line.hには、少なくともget_next_line()関数のプロトタイプが含まれていなければならない。

- Your header file `get_next_line.h` must at least contain the prototype of the `get_next_line()` function.

get_next_line_utils.cファイルに必要なヘルパー関数をすべて追加する。

- Add all the helper functions you need in the `get_next_line_utils.c` file.

手始めに、静的変数とは何かを知っておくといいだろう。
A good start would be to know what a static variable is.

get_next_line()でファイルを読み込む必要があるため、次のオプションをコンパイラ呼び出しに追加： -D BUFFER_SIZE=n

- Because you will have to read files in `get_next_line()`, add this option to your compiler call: `-D BUFFER_SIZE=n`
  It will define the buffer size for `read()`. これはread()のバッファサイズを定義する。
  The buffer size value will be modified by your peer-evaluators and the Moulinette in order to test your code. バッファサイズの値はレビュー時に変更される。

通常のフラグに加えて、-D BUFFER_SIZEフラグの有無でこのプロジェクトをコンパイルできなければならない。
お好みのデフォルト値を選んでください。

We must be able to compile this project with and without the -D BUFFER_SIZE flag in addition to the usual flags. You can choose the default value of your choice.

以下のようにコンパイル（例としてバッファサイズ42を使用）：cc -Wall -Wextra -Werror -D BUFFER_SIZE=42 <files>.c

- You will compile your code as follows (a buffer size of 42 is used as an example):
  `cc -Wall -Wextra -Werror -D BUFFER_SIZE=42 <files>.c`
- We consider that `get_next_line()` has an undefined behavior if the file pointed to by the file descriptor changed since the last call whereas `read()` didn't reach the end of file. get_next_line()は、read()がファイルの末尾に到達しなかった最後の呼び出し以降に、ファイル記述子が指すファイルが変更された場合、未定義の動作をすると考える。
- We also consider that `get_next_line()` has an undefined behavior when reading a binary file. However, you can implement a logical way to handle this behavior if you want to. また、get_next_line()はバイナリファイルを読み込む際に未定義の動作をすることも考慮する。しかし、もし必要なら、この動作を処理する論理的な方法を実装することができる。

Does your function still work if the BUFFER_SIZE value is 9999? If it is 1? 10000000? Do you know why?
BUFFER_SIZEが9999や1、10000000?の場合でも、この関数は動作しますか？

get_next_line()が呼び出されるたびに、できるだけ読み込まないようにする。改行があったら、現在の行を返さなければならない。ファイル全体を読んでから各行を処理するのはやめましょう。

Try to read as little as possible each time get_next_line() is called. If you encounter a new line, you have to return the current line.
Don't read the whole file and then process each line.

Forbidden

- You are not allowed to use your `libft` in this project.
- `lseek()` is forbidden.
- Global variables are forbidden.

Chapter IV

Bonus part

This project is straightforward and doesn't allow complex bonuses. However, we trust your creativity. If you completed the mandatory part, give a try to this bonus part.

Here are the bonus part requirements:

- Develop `get_next_line()` using only one static variable.
- Your `get_next_line()` can manage multiple file descriptors at the same time.
  For example, if you can read from the file descriptors 3, 4 and 5, you should be able to read from a different fd per call without losing the reading thread of each file descriptor or returning a line from another fd.
  It means that you should be able to call `get_next_line()` to read from fd 3, then fd 4, then 5, then once again 3, once again 4, and so forth.

Append the `_bonus.[c\h]` suffix to the bonus part files.
It means that, in addition to the mandatory part files, you will turn in the 3 following files:

- `get_next_line_bonus.c`
- `get_next_line_bonus.h`
- `get_next_line_utils_bonus.c`

⚠ The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter V

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.

```
When writing your tests, remember that:
1) Both the buffer size and the line size can be of very different
values.
2) A file descriptor does not only point to regular files.
Be smart and cross-check with your peers. Prepare a full set of
diverse tests for defense.
```

Once passed, do not hesitate to add your `get_next_line()` to your `libft`.

テストを書く際には、次のことを覚えておいてほしい：
1) バッファ・サイズも行サイズも、非常に異なる値になりうる。
2) ファイル記述子は通常のファイルだけを指すわけではない。
賢く、仲間とクロスチェックしてください。防衛のために、多様なテストのフルセットを準備すること。

```
/=∂/\/\[](_)§ /\/\@|V †|-|@¯|¯ /-/!570®1<|-\£1_`/ ¢@/\/\ε vv!7}{ ???
```