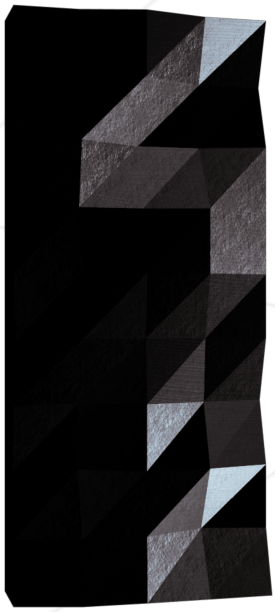

Get Next Line

Reading a line from a fd is way too tedious

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

Version: 11

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.

Ce projet vous permettra non seulement d'ajouter une fonction très pratique à votre collection, mais aussi d'apprendre un nouveau concept très intéressant de la programmation en C : les variables statiques.

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, librairies 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.

- Votre projet doit être écrit en C.
- Votre projet doit être écrit conformément à la norme. Si vous avez des fichiers/fonctions bonus, ils sont inclus dans la vérification de la norme et vous recevrez un 0 s'il y a une erreur de norme à l'intérieur.
- Vos fonctions ne doivent pas se terminer de manière inattendue (erreur de segmentation, erreur de bus, double free, etc) en dehors des comportements non définis. Si cela se produit, votre projet sera considéré comme non fonctionnel et recevra un 0 lors de l'évaluation.
- Tout l'espace mémoire alloué au tas doit être correctement libéré lorsque cela est nécessaire. Aucune fuite ne sera tolérée.
- Si le sujet l'exige, vous devez soumettre un Makefile qui compilera vos fichiers sources vers la sortie requise avec les drapeaux -Wall, -Wextra et -Werror, utiliser cc, et votre Makefile ne doit pas être relinké.
- Votre Makefile doit au moins contenir les règles $(NAME), all, clean, fclean et re.
- Pour ajouter des bonus à votre projet, vous devez inclure une règle bonus à votre Makefile, qui ajoutera tous les différents en-têtes, librairies ou fonctions qui sont interdits dans la partie principale du projet. Les bonus doivent être dans un fichier différent _bonus.{c/h} si le sujet ne spécifie rien d'autre. L'évaluation de la partie obligatoire et de la partie bonus se fait séparément.
- Si votre projet vous permet d'utiliser votre libft, vous devez copier ses sources et son Makefile associé dans un dossier libft avec son Makefile associé. Le Makefile de votre projet doit compiler la bibliothèque en utilisant son Makefile, puis compiler le projet.
- Nous vous encourageons à créer des programmes de test pour votre projet, même si ce travail ne devra pas être soumis et ne sera pas noté. Cela vous permettra de tester facilement votre travail et celui de vos camarades. Ces tests vous seront particulièrement utiles lors de votre soutenance. En effet, lors de la soutenance, vous êtes libre d'utiliser vos tests et/ou les tests du pair que vous évaluez.
- Soumettez votre travail au dépôt git qui vous a été attribué. Seul le travail dans le dépôt git sera noté. Si Deepthought est chargé de noter votre travail, il le fera après vos évaluations par les pairs. Si une erreur survient dans une section de votre travail pendant l'évaluation de Deepthought, l'évaluation sera interrompue.

Chapter III

Mandatory part

- Des appels répétés (par exemple, à l'aide d'une boucle) à la fonction get_next_line() doivent vous permettre de lire le fichier texte indiqué par le descripteur de fichier, une ligne à la fois.
- Votre fonction doit renvoyer la ligne qui a été lue.
S'il n'y a rien d'autre à lire ou si une erreur s'est produite, elle doit renvoyer NULL.
- Assurez-vous que votre fonction fonctionne comme prévu à la fois lors de la lecture d'un fichier et lors de la lecture à partir de l'entrée standard.
- Veuillez noter que la ligne renvoyée doit inclure le caractère de fin de fichier, sauf si la fin du fichier a été atteinte et qu'elle ne se termine pas par un caractère de fin de fichier.
- Votre fichier d'en-tête get_next_line.h doit au moins contenir le prototype de la fonction get_next_line().
- Ajoutez toutes les fonctions d'aide dont vous avez besoin dans le fichier get_next_line_utils.c.

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`
Return value	`Read line: correct behavior` `NULL: there is nothing else to read, or an error occurred`
External functs.	`read, malloc, free`
Description	`Write a function that returns a line read from a file descriptor`

- 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**.
- 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`.
- Make sure that your function works as expected both when reading a file and when reading from the standard input.
- **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.
- Your header file `get_next_line.h` must at least contain the prototype of the `get_next_line()` function.
- Add all the helper functions you need in the `get_next_line_utils.c` file.

A good start would be to know what a static variable is.

- 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()`.
  The buffer size value will be modified by your peer-evaluators and the Moulinette in order to test your code.

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.

- 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.
- 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.

Does your function still work if the BUFFER_SIZE value is 9999? If it is 1? 10000000? Do you know why?

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.

- Comme vous devrez lire des fichiers dans get_next_line(), ajoutez cette option à votre appel au compilateur : -D BUFFER_SIZE=n
Elle définira la taille de la mémoire tampon pour read().
La valeur de la taille du tampon sera modifiée par vos pairs évaluateurs et la Moulinette afin de tester votre code.
- Vous compilerez votre code comme suit (une taille de tampon de 42 est utilisée à titre d'exemple) : cc -Wall -Wextra -Werror -D BUFFER_SIZE=42 <files>.c
- Nous considérons que get_next_line() a un comportement indéfini si le fichier pointé par le descripteur de fichier a changé depuis le dernier appel alors que read() n'a pas atteint la fin du fichier.
- Nous considérons également que get_next_line() a un comportement indéfini lors de la lecture d'un fichier binaire. Cependant, vous pouvez implémenter une manière logique de gérer ce comportement si vous le souhaitez.

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.

Ce projet est simple et ne permet pas de bonus complexes. Cependant, nous faisons confiance à votre créativité. Si vous avez terminé la partie obligatoire, essayez cette partie bonus.
Voici les exigences de la partie bonus :
- Développez get_next_line() en utilisant une seule variable statique.
- Votre get_next_line() peut gérer plusieurs descripteurs de fichiers en même temps. Par exemple, si vous pouvez lire à partir des descripteurs de fichiers 3, 4 et 5, vous devriez pouvoir lire à partir d'un fd différent par appel sans perdre le thread de lecture de chaque descripteur de fichier ou renvoyer une ligne à partir d'un autre fd.
Cela signifie que vous devriez pouvoir appeler get_next_line() pour lire à partir du fd 3, puis du fd 4, puis du fd 5, puis à nouveau du fd 3, à nouveau du fd 4, et ainsi de suite.
Ajoutez le suffixe _bonus.[c\h] aux fichiers de parties bonus.

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`.