



## Get Next Line

Parce que lire sur un fd c'est pas passionnant

*Résumé: Ce projet a pour but de vous faire coder une fonction qui renvoie une ligne terminée par un retour à la ligne lue depuis un descripteur de fichier.*

# Table des matières

<b>I</b>	<b>Règles communes</b>	<b>2</b>
<b>II</b>	<b>Objectifs</b>	<b>3</b>
<b>III</b>	<b>Partie Obligatoire - Get_next_line</b>	<b>4</b>
<b>IV</b>	<b>Partie Bonus</b>	<b>6</b>

# Chapitre I

## Règles communes

- Votre projet doit être codé à la Norme. Si vous avez des fichiers ou fonctions bonus, celles-ci seront incluses dans la vérification de la norme et vous aurez 0 au projet en cas de faute de norme.
- Vos fonctions ne doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Toute mémoire allouée sur la heap doit être libérée lorsque c'est nécessaire. Aucun leak ne sera toléré.
- Si le projet le demande, vous devez rendre un Makefile qui compilera vos sources pour créer la sortie demandée, en utilisant les flags `-Wall`, `-Wextra` et `-Werror`. Votre Makefile ne doit pas relink.
- Si le projet demande un Makefile, votre Makefile doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Pour rendre des bonus, vous devez inclure une règle `bonus` à votre Makefile qui ajoutera les divers headers, librairies ou fonctions qui ne sont pas autorisées dans la partie principale du projet. Les bonus doivent être dans un fichier `_bonus.{c/h}`. L'évaluation de la partie obligatoire et de la partie bonus sont faites séparément.
- Si le projet autorise votre `libft`, vous devez copier ses sources et son Makefile associé dans un dossier `libft` contenu à la racine. Le Makefile de votre projet doit compiler la librairie à l'aide de son Makefile, puis compiler le projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

# Chapitre II

## Objectifs

Ce projet va non seulement vous permettre d'ajouter une fonction très pratique à votre collection, mais vous permettra également d'aborder un nouvel élément surprenant de la programmation en C : les variables statiques.

# Chapitre III

## Partie Obligatoire - Get\_next\_line

Function name	
Prototype	<code>int get_next_line(int fd, char **line);</code>
Fichiers de rendu	<code>get_next_line.c</code> , <code>get_next_line_utils.c</code> , <code>get_next_line.h</code>
Paramètres	#1. le file descriptor sur lequel lire #2. La valeur de ce qui a été lu
Valeur de retour	1 : Une ligne a été lue 0 : La fin de fichier a été atteinte -1 : Une erreur est survenue
Fonctions externes autorisées	<code>read</code> , <code>malloc</code> , <code>free</code>
Description	Ecrivez une fonction qui retourne une ligne lue depuis un file descriptor, sans le retour à la ligne

- Des appels successifs à votre fonction `get_next_line` doivent vous permettre de lire l'entièreté du texte disponible sur le file descriptor, une ligne à la fois, jusqu'au EOF.
- La `libft` n'est pas autorisée sur ce projet. Vous devez ajouter le fichier `get_next_line_utils.c` qui contiendra les fonctions nécessaires au fonctionnement de votre `get_next_line`.
- Assurez-vous que votre fonction se comporte correctement lorsque vous lisez depuis un fichier ou l'entrée standard.
- Votre programme doit compiler avec le flag `-D BUFFER_SIZE=xx` Ce define doit être utilisé dans vos appels de `read` du `get_next_line`, pour définir la taille du buffer. Cette valeur sera modifiée lors de l'évaluation et par la moulinette.
- Compilation : `gcc -Wall -Wextra -Werror -D BUFFER_SIZE=32 get_next_line.c get_next_line_utils.c`
- Votre `read` DOIT utiliser le `BUFFER_SIZE` pour lire depuis un fichier ou depuis le `stdin`.
- Dans le fichier header `get_next_line.h`, vous devez avoir au moins le prototype

de la fonction.



Votre fonction fonctionne-t-elle encore si la valeur de `BUFFER_SIZE` est 9999 ? Et si elle vaut 1 ? ou 10000000 ? Savez-vous pourquoi ?



Votre programme doit lire le moins possible à chaque fois que `get_next_line` est appelé. Si vous rencontrez un retour à la ligne, vous devez renvoyer la ligne et arrêter de lire jusqu'au prochain appel. Évitez de tout lire d'abord puis de travailler les lignes une fois que tout est lue.



Ne rendez pas votre projet sans tester. Il y a beaucoup de tests à faire, portez-y attention. Essayez de lire depuis un fichier, depuis une redirection, depuis l'entrée standard. Comment se comporte votre programme lorsque vous lui envoyez un retour à la ligne ? Ou CTRL-D ?

- Nous considérons que `get_next_line` a un comportement indéterminé si, entre deux appels, le file descriptor change de fichier alors qu'EOF n'a pas été atteint sur le premier fichier.
- Non, `lseek` n'est pas une fonction autorisée. La lecture du file descriptor ne doit être faite qu'une seule fois.
- Enfin, nous considérons que `get_next_line` a un comportement indéterminé si nous lisons un fichier binaire. Cependant, si vous le souhaitez, vous pouvez rendre ce comportement cohérent.
- Les variables globales sont interdites.



Savoir ce qu'est une variable statique est un bon point de départ.

[https://en.wikipedia.org/wiki/Static\\_variable](https://en.wikipedia.org/wiki/Static_variable)

# Chapitre IV

## Partie Bonus

Le projet `get_next_line` laisse peu de place à l'imagination, mais si vous avez complété entièrement la partie obligatoire, vous pouvez faire les bonus proposés ici. Rendez les 3 fichiers initiaux `_bonus` pour cette partie.

- Completez `get_next_line` avec une seule variable statique.
- Completez `get_next_line` en lui permettant de gérer plusieurs fd. Par exemple, si les fd 3, 4 et 5 sont accessibles en lecture, alors vous pouvez appeler `get_next_line` une fois sur 3, puis sur 4, puis sur 5, puis le rappeler sur 3, etc. Sans jamais perdre le contenu lu sur chacun des fd, et sans le mélanger.