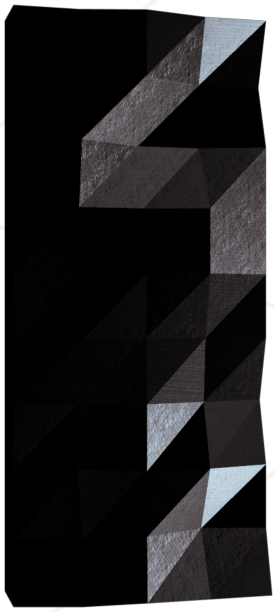

Get Next Line

Parce que lire depuis un fd, c'est pas passionnant

Résumé:
Ce projet a pour but de vous faire développer une fonction qui renvoie une ligne lue depuis un descripteur de fichier.

Version: 12

Table des matières

Chapitre I

Objectifs

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

Chapitre II

Règles communes

- Votre projet doit être écrit en C.
- Votre projet doit être codé à la Norme. Si vous avez des fichiers ou fonctions bonus, celles-ci seront inclues 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 libéré 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 différent : `_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 III

Partie obligatoire

Function name	`get_next_line`
Prototype	`char *get_next_line(int fd);`
Fichiers de rendu	`get_next_line.h, get_next_line.c, get_next_line_utils.c`
Paramètres	`fd : le descripteur de fichier depuis lequel lire`
Valeur de retour	`Le contenu de la ligne lue : comportement correct NULL : rien d'autre à lire ou une erreur s'est produite`
Fonctions externes autorisées	`read, malloc, free`
Description	`Écrire une fonction qui retourne une ligne lue depuis un descripteur de fichier`

- Des appels successifs à votre fonction `get_next_line()` doivent vous permettre de lire l'intégralité du fichier texte référencé par le descripteur de fichier, **une ligne à la fois**.
- Votre fonction doit retourner la ligne qui vient d'être lue.
  S'il n'y a plus rien à lire, ou en cas d'erreur, elle doit retourner `NULL`.
- Assurez-vous que votre fonction se comporte correctement qu'elle lise un fichier ou qu'elle lise sur l'entrée standard.
- **Important :** Vous devez toujours retourner la ligne qui a été lue suivie du `\n` la terminant, sauf dans le cas où vous avez atteint la fin du fichier et que ce dernier ne se termine pas par un `\n`.
- Le fichier d'en-tête `get_next_line.h` doit contenir au minimum le prototype de la fonction.
- Le fichier `get_next_line_utils.c` vous servira à ajouter des fonctions supplémentaires nécessaires à la réalisation de votre `get_next_line()`.

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

- Votre programme doit compiler avec l'option : `-D BUFFER_SIZE=n`
  Cette macro définie à l'invocation du compilateur servira à spécifier la taille du *buffer* lors de vos appels à `read()` dans votre fonction `get_next_line()`.
  Cette valeur sera modifiée lors de la peer-evaluation et par la Moulinette dans le but de tester votre rendu.

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.

- Votre programme sera donc compilé de la manière suivante (exemple ci-dessous avec une taille de *buffer* de 42) :
  `cc -Wall -Wextra -Werror -D BUFFER_SIZE=42 <files>.c`

- Nous considérons que `get_next_line()` a un comportement indeterminé si, entre deux appels, le fichier pointé par le descripteur de fichier a été modifié, alors que le premier fichier n'a pas été lu en entier.

- Nous considérons aussi que `get_next_line()` a un comportement indeterminé en cas de lecture d'un fichier binaire. Cependant, si vous le souhaitez, vous pouvez rendre ce comportement cohérent.

Votre fonction marche-t-elle encore si la valeur de BUFFER_SIZE est de 9999? Ou de 1 ? Ou encore de 10 000 000 ? Savez-vous pourquoi ?

Votre programme doit lire le moins possible à chaque appel à get_next_line(). Si vous rencontrez une nouvelle ligne, vous devez retourner la ligne précédente venant d'être lue.
Ne lisez pas l'intégralité du fichier pour ensuite traiter chaque ligne.

Ce qui n'est pas autorisé

- La `libft` n'est pas autorisée pour ce projet.
- La fonction `lseek()` est interdite.
- Les variables globales sont interdites.

Chapitre IV

Partie Bonus

Du fait de sa simplicité, le projet `get_next_line` laisse peu de place aux bonus, mais nous sommes sûrs que vous avez beaucoup d'imagination. Si vous avez réussi la partie obligatoire, alors n'hésitez pas à faire les bonus pour aller plus loin.

Voici les bonus à réaliser :

- Faites `get_next_line()` avec une seule variable statique.
- Complétez `get_next_line()` en lui permettant de gérer plusieurs fd.
  Par exemple, si les fd 3, 4 et 5 sont accessibles en lecture, alors il est possible de les lire chacun leur tour sans jamais perdre les contenus lus sur chacun des fd, et sans retourner la mauvaise ligne.
  Vous devriez pouvoir appeler `get_next_line()` une fois avec le fd 3, puis le 4, le 5, puis à nouveau le 3, à nouveau le 4, etc.

Ajoutez le suffixe `_bonus.[c\h]` aux fichiers de cette partie.
Ainsi, en plus des 3 fichiers de la partie obligatoire, vous rendrez les 3 fichiers suivants :

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

Les bonus ne seront évalués que si la partie obligatoire est PARFAITE. Par parfaite, nous entendons complète et sans aucun dysfonctionnement. Si vous n'avez pas réussi TOUS les points de la partie obligatoire, votre partie bonus ne sera pas prise en compte.

Chapitre V

Rendu et peer-evaluation

Rendez votre travail sur votre dépot `Git` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Pour vos tests, gardez à l'esprit que :
1) Le buffer et la ligne peuvent être de tailles très différentes.
2) Un descripteur de fichier ne pointe pas seulement sur de simples fichiers.
Soyez malins et vérifiez votre travail avec vos pairs. Préparez un large éventail de tests pour votre évaluation.

Une fois ce projet validé, n'hésitez pas à ajouter votre `get_next_line()` à votre `libft`.

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