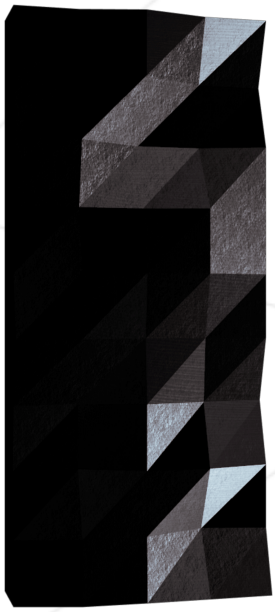

Minishell

Aussi mignon qu'un vrai shell

Résumé:
L'objectif de ce projet est de créer un shell minimaliste.
Ça sera votre petit bash à vous.
Vous en apprendrez beaucoup sur les processus et les descripteurs de fichier.

Version: 6

Table des matières

Chapitre I

Introduction

L'existence des shells est intrinsèquement liée à celle de l'informatique.

À l'époque, les développeurs étaient tous d'accord pour dire que *communiquer avec un ordinateur en utilisant des interrupteurs 1/0 était fortement irritant.*

La suite logique fut d'inventer un moyen de communiquer via des lignes de commandes interactives dans un langage jusqu'à un certain point proche du langage humain.

Avec `Minishell`, vous allez voyager dans le passé et faire face aux problèmes que l'on pouvait avoir au temps où *Windows* n'existait pas encore.

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 de 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

Nom du programme	`minishell`
Fichiers de rendu	`Makefile, *.h, *.c`
Makefile	`NAME, all, clean, fclean, re`
Arguments	
Fonctions externes autorisées	`readline`, `rl_clear_history`, `rl_on_new_line`, `rl_replace_line`, `rl_redisplay`, `add_history`, `printf`, `malloc`, `free`, `write`, `access`, `open`, `read`, `close`, `fork`, `wait`, `waitpid`, `wait3`, `wait4`, `signal`, `sigaction`, `sigemptyset`, `sigaddset`, `kill`, `exit`, `getcwd`, `chdir`, `stat`, `lstat`, `fstat`, `unlink`, `execve`, `dup`, `dup2`, `pipe`, `opendir`, `readdir`, `closedir`, `strerror`, `perror`, `isatty`, `ttyname`, `ttyslot`, `ioctl`, `getenv`, `tcsetattr`, `tcgetattr`, `tgetent`, `tgetflag`, `tgetnum`, `tgetstr`, `tgoto`, `tputs`
Libft autorisée	`Oui`
Description	`Développez un shell`

Votre shell doit :

- Afficher un **prompt** en l'attente d'une nouvelle commande.
- Posséder un **historique** fonctionnel.
- Chercher et lancer le bon exécutable (en se basant sur la variable d'environnement `PATH`, ou sur un chemin relatif ou absolu).
- Ne pas utiliser **plus d'une variable globale**. Réfléchissez-y car vous devrez justifier son utilisation.
- Ne pas interpréter de *quotes* (guillemets) non fermés ou de caractères spéciaux non demandés dans le sujet, tels que `\` (le *backslash*) ou `;` (le point-virgule).
- Gérer `'` (*single quote*) qui doit empêcher le shell d'interpréter les méta-caractères présents dans la séquence entre guillemets.
- Gérer `"` (*double quote*) qui doit empêcher le shell d'interpréter les méta-caractères présents dans la séquence entre guillemets sauf le $ (signe dollar).

- Implémenter les **redirections** :
  - `<` doit rediriger l'entrée.
  - `>` doit rediriger la sortie.
  - `<<` doit recevoir un délimiteur et lire l'input donné jusqu'à rencontrer une ligne contenant le délimiteur. Cependant, l'historique n'a pas à être mis à jour !
  - `>>` doit rediriger la sortie en mode *append*.
- Implémenter les **pipes** (caractère `|`). La sortie de chaque commande de la *pipeline* est connectée à l'entrée de la commande suivante grâce à un *pipe*.
- Gérer les **variables d'environnement** (un `$` suivi d'une séquence de caractères) qui doivent être substituées par leur contenu.
- Gérer `$?` qui doit être substitué par le statut de sortie de la dernière *pipeline* exécutée au premier plan.
- Gérer `ctrl-C`, `ctrl-D` et `ctrl-\` qui doivent fonctionner comme dans `bash`.
- En mode interactif :
  - `ctrl-C` affiche un nouveau *prompt* sur une nouvelle ligne.
  - `ctrl-D` quitte le shell.
  - `ctrl-\` ne fait rien.
- Votre shell doit implémenter les **builtins** suivantes :
  - `echo` et l'option `-n`
  - `cd` uniquement avec un chemin relatif ou absolu
  - `pwd` sans aucune option
  - `export` sans aucune option
  - `unset` sans aucune option
  - `env` sans aucune option ni argument
  - `exit` sans aucune option

La fonction `readline()` peut causer des fuites de mémoire. Vous n'avez pas à les gérer. Attention, cela **ne veut pas pour autant dire que votre code, oui celui que vous avez écrit, peut avoir des fuites de mémoire**.

Tenez-vous en à ce qui est demandé dans le sujet. Ce qui n'est pas demandé n'est pas obligatoire.
Si vous avez un doute sur une consigne du sujet, prenez bash comme référence.

Chapitre IV

Partie bonus

Votre programme doit implémenter :

- `&&` et `||` avec des parenthèses pour les priorités.
- Les *wildcards* `*` doivent fonctionner pour le répertoire courant.

`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.