Minishell

Aussi beau qu’un shell

Résumé :

Ce projet consiste à créer un shell simple.

Oui, votre propre petit bash.
Vous apprendrez beaucoup sur les processus et les descripteurs de fichiers.

Version : 7.1

Sommaire :

Chapitre I

Introduction

L'existence des shells est liée à celle de l'informatique elle-même.

À l'époque, tous les développeurs étaient d'accord : communiquer avec un ordinateur à l'aide d'interrupteurs 1/0 alignés était extrêmement irritant.

Il était donc logique d'imaginer un logiciel permettant de communiquer avec un ordinateur en utilisant des lignes de commande interactives dans un langage quelque peu proche du langage humain.

Grâce à Minishell, vous pourrez voyager dans le temps et revivre les problèmes auxquels les développeurs faisaient face avant l'existence de Windows.

Chapitre II

Instructions générales

- Votre projet doit être écrit en C.

- Votre projet doit respecter la Norme. Si vous avez des fichiers ou fonctions bonus, ils seront inclus dans la vérification de la norme, et vous obtiendrez un 0 s'il y a une erreur de norme.
- Vos fonctions ne doivent pas quitter de manière inattendue (segfault, bus error, double free, etc.), sauf en cas de comportements indéfinis. Si cela se produit, votre projet sera considéré comme non fonctionnel et recevra un 0.
- Toute mémoire allouée sur le tas (heap) doit être correctement libérée lorsque nécessaire. Les fuites de mémoire ne seront pas tolérées.
- Si le sujet le demande, vous devez soumettre un Makefile qui compilera vos fichiers source avec les options `-Wall`, `-Wextra` et `-Werror`, en utilisant `cc`. Votre Makefile ne doit pas relinker.
- Votre Makefile doit au moins contenir les règles suivantes : `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Pour inclure des bonus, ajoutez une règle `bonus` à votre Makefile, qui inclura les headers, bibliothèques ou fonctions interdites dans la partie obligatoire. Les fichiers bonus doivent être dans des fichiers séparés `*_bonus.{c/h}`, sauf mention contraire. Les évaluations des parties obligatoire et bonus se font séparément.
- Si vous pouvez utiliser votre libft, copiez ses sources et son Makefile dans un dossier `libft`. Le Makefile principal devra compiler cette bibliothèque avant de compiler le projet.
- Nous vous encourageons à créer des programmes de test, même si ces derniers ne seront pas soumis ni évalués. Ces tests seront particulièrement utiles lors de votre défense. Pendant la défense, vous êtes libre d'utiliser vos propres tests ou ceux de vos pairs.
- Soumettez votre travail dans le dépôt Git assigné. Seul le contenu de ce dépôt sera évalué. Si Deepthought évalue votre travail, toute erreur arrêtera l'évaluation.

Chapitre III

Partie obligatoire

- **Nom du programme :** `minishell`
- **Fichiers à rendre :** `Makefile, *.h, *.c`
- **Règles Makefile :** `NAME, all, clean, fclean, re`
- **Arguments :** aucun
- **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 :**
  Créez un shell capable de :
  - Afficher un prompt en attente d'une commande.
  - Gérer un historique fonctionnel.
  - Rechercher et exécuter le bon exécutable (basé sur `$PATH` ou via un chemin relatif/absolu).
  - Gérer les redirections (`<`, `>`, `<<`, `>>`).
  - Gérer les pipes (`|`).
  - Gérer les variables d'environnement (`$VAR`).
  - Implémenter les builtins :
    - `echo` (avec option `-n`),
    - `cd` (avec chemins relatifs ou absolus),
    - `pwd`,
    - `export`,
    - `unset`,
    - `env`,
    - `exit`.
  - Gérer les signaux (`ctrl-C`, `ctrl-D`, `ctrl-\`) comme dans Bash.

Les fuites de mémoire dans `readline()` ne sont pas votre responsabilité, mais votre propre code doit être exempt de fuites.

Chapitre IV

Partie bonus

Votre programme doit implémenter :

- Les opérateurs logiques `&&` et `||` avec priorités via des parenthèses.
- Les wildcards (`*`) dans le répertoire courant.

Les bonus ne seront évalués que si la partie obligatoire est **parfaite** (sans défaut).

Chapitre V

Soumission et évaluation

Soumettez votre projet dans le dépôt Git assigné. Vérifiez les noms de vos fichiers pour éviter les erreurs.