Chapitre I

Introduction

L’existence des shells est liée à celle de l’informatique.
À l'époque, tous les développeurs s'accordaient à dire que communiquer avec un ordinateur en utilisant des interrupteurs alignés 1/0 était extrêmement agaçant.
Il était donc logique qu’ils aient eu l’idée de créer un logiciel permettant de communiquer avec un ordinateur en utilisant des lignes de commandes interactives dans un langage relativement proche du langage humain.
Grâce à Minishell, vous pourrez voyager dans le temps et revenir aux problèmes rencontrés lorsque Windows n’existait pas.

Chapitre II

Instructions communes

- Votre projet doit être écrit en C.
- Votre projet doit être conforme à la Norme. Si vous avez des fichiers ou fonctions bonus, ils sont inclus dans la vérification de la norme, et vous recevrez un 0 s'il y a une erreur de norme.
- Vos fonctions ne doivent pas se terminer de manière inattendue (segmentation fault, bus error, double free, etc.), excepté dans le cas de comportements indéfinis. Si cela arrive, votre projet sera considéré comme non fonctionnel et recevra un 0 lors de l'évaluation.
- Toute mémoire allouée sur le tas (heap) doit être correctement libérée si nécessaire. Aucune fuite de mémoire ne sera tolérée.
- Si le sujet le demande, vous devez soumettre un Makefile qui compilera vos fichiers source avec les flags `-Wall`, `-Wextra` et `-Werror`, en utilisant `cc`. Votre Makefile ne doit pas relinker.
- Votre Makefile doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean`, et `re`.
- Pour inclure des bonus dans votre projet, vous devez inclure une règle `bonus` dans votre Makefile, qui ajoutera les différents fichiers headers, bibliothèques ou fonctions interdits dans la partie principale du projet. Les bonus doivent être dans des fichiers distincts _bonus. {c/h}, sauf si le sujet précise autre chose. L’évaluation des parties obligatoire et bonus est faite séparément.
- Si votre projet vous autorise à utiliser votre libft, vous devez copier ses sources et son Makefile dans un dossier `libft`. 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 n'a pas à être soumis et ne sera pas noté. Cela vous donnera l'opportunité de tester facilement votre travail et celui de vos pairs. 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 ceux de la personne que vous évaluez.
- Soumettez votre travail dans le dépôt git qui vous est assigné. Seul le travail dans le dépôt git sera évalué. Si Deepthought est chargé de noter votre travail, cela se fera après les

évaluations par vos pairs. Si une erreur survient dans une section de votre travail pendant la notation de Deepthought, l'évaluation s'arrêtera.

Chapitre III

Partie obligatoire

Nom du programme : minishell
Fichiers à rendre : Makefile, *.h, *.c
Règles du Makefile : `NAME`, `all`, `clean`, `fclean`, `re`
Arguments :
Fonctions externes : `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 : Écrire un shell.

Votre shell doit :

- Afficher un prompt en attente d’une nouvelle commande.
- Avoir un historique fonctionnel.
- Rechercher et lancer le bon exécutable (basé sur la variable PATH ou en utilisant un chemin relatif ou absolu).
- Ne pas utiliser plus d'une variable globale pour indiquer un signal reçu. Cela garantit que votre gestionnaire de signaux n'accèdera pas à vos structures de données principales. Cette variable globale ne peut fournir aucune autre information que le numéro du signal reçu.
- Ne pas interpréter les guillemets non fermés ou les caractères spéciaux qui ne sont pas requis par le sujet, tels que \ (backslash) ou ; (point-virgule).
- Gérer les guillemets simples ('), empêchant le shell d’interpréter les métacaractères dans la séquence citée.
- Gérer les guillemets doubles ("), empêchant le shell d’interpréter les métacaractères sauf pour $ (signe dollar).
- Implémenter les redirections :
  - < pour rediriger l’entrée,
  - > pour rediriger la sortie,
  - << pour lire l'entrée jusqu'à rencontrer un délimiteur, sans mettre à jour l'historique,
  - >> pour rediriger la sortie en mode append.
- Implémenter les pipes (|), où la sortie de chaque commande est reliée à l’entrée de la suivante via un pipe.
- Gérer les variables d'environnement ($ suivi d'une séquence de caractères), qui doivent être étendues à leur valeur.

- Gérer `?`, qui doit s'étendre au statut de sortie de la dernière commande au premier plan exécutée.
- Gérer `ctrl-C`, `ctrl-D` et `ctrl-\` comme dans bash :
  - `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 suivants :
  - `echo` avec l'option `-n`,
  - `cd` avec un chemin relatif ou absolu,
  - `pwd` sans options,
  - `export` sans options,
  - `unset` sans options,
  - `env` sans options ni arguments,
  - `exit` sans options.

La fonction `readline()` peut provoquer des fuites de mémoire. Vous n’avez pas à les corriger. Mais cela ne signifie pas que votre propre code peut en avoir.
Limitez-vous à la description du sujet. Tout ce qui n’est pas demandé n’est pas requis. En cas de doute, 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.

La partie bonus sera évaluée uniquement si la partie obligatoire est PARFAITE, c'est-à-dire si elle est intégralement réalisée et fonctionne sans dysfonctionnement. Si vous n'avez pas rempli toutes les exigences obligatoires, la partie bonus ne sera pas évaluée.