

### Résumé

Le but de cette série de TP est d'écrire un mini interpréteur de commandes (shell). Son nom sera `iutsh`. Nous commençons par un interpréteur basique puis, au cours des TP suivants, nous lui ajouterons des fonctionnalités de plus en plus évoluées.

**N'oubliez donc pas de sauvegarder chaque version intermédiaire qui fonctionne avant de la modifier !!!**

## Présentation

On rappelle qu'«un langage de commande (shell) est un programme capable d'interpréter des commandes qui seront exécutées par le système d'exploitation». L'algorithme de base d'un tel programme est le suivant :

1. Afficher une invite de commande (prompt).
2. Lire une ligne de commandes.
3. Interpréter et exécuter les commandes lues.
4. Retourner en 1.

Pour passer de l'étape 3 à l'étape 4 le shell doit attendre la fin de l'exécution des commandes démarrées à l'étape 3 sauf si l'exécution de ces commandes a été demandée en tâche de fond ou arrière plan (si la commande est suivie du caractère «&»).

L'étape 1 sera effectuée par la fonction :

```
void affiche_prompt();
```

L'étape 2 sera effectuée par la fonction :

```
char *** ligne_commande(int * flag, int * nb);
```

L'étape 3 sera effectuée par la fonction :

```
void execute_ligne_commande();
```

Le but de ces TP n'étant pas de manipuler des chaînes de caractères ou de faire de l'analyse syntaxique, la fonction `ligne_commande` et quelques fonctions annexes vous sont données. Le mode d'emploi de ces fonctions est spécifié en fin d'énoncé. Il vous suffira donc d'intégrer l'appel de la fonction `ligne_commande` à la fonction `execute_ligne_commande`.

### Exercice 1 : Exécution d'une commande

Écrire la fonction

```
void execute_ligne_commande();
```

Elle doit lire une commande et l'exécuter. La lecture de cette ligne se fait grâce à la fonction `ligne_commande` décrite en fin d'énoncé. Pour exécuter la commande, vous devez utiliser la fonction `execvp`. Lorsque la fonction `execute_ligne_commande` est au point, vous devez y ajouter les traitements nécessaires pour prendre en compte les exécutions en tâche de fond (gestion du caractère «&» en fin de ligne).

### Exercice 2 : L'interpréteur de base

Vous pouvez maintenant écrire la première version de votre interpréteur de commande. Pour le moment la fonction `affiche_prompt` affiche simplement le prompt «`iutsh$`».

### Exercice 3 : Invite de commande évoluée (prompt)

Vous allez à présent modifier le prompt. Modifier pour cela la fonction

```
void affiche_prompt();
```

Elle doit permettre d'afficher un prompt constitué :

- du nom de l'utilisateur ;
- du nom de la machine ;
- du répertoire courant.

Vous devez pour cela utiliser les fonctions `getenv`, `gethostname` et `getcwd`. Lisez attentivement les pages du manuel de ces fonctions afin de les utiliser correctement. Faites, en particulier, attention à ce qui concerne la taille des chaînes de caractères. Pour améliorer la lisibilité du prompt vous pouvez ajouter les accolades ou séparateurs qui vous plaisent.

## Lecture de la ligne de commandes

Le but de cette série de TP n'étant pas la manipulation de chaînes de caractères, les fonctions de saisie d'une ligne de commandes ainsi que quelques fonctions annexes vous sont fournies. Vous trouverez dans le répertoire `~hauspiem/public/asr4` les fichiers `ligne_commande.h` et `ligne_commande.o`.

- La fonction

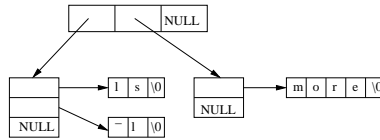
```
char *** ligne_commande (int * flag, int * nb);
```

lit une ligne de commandes et construit un tableau de tableaux de chaînes de caractères. Chaque commande lue est transformée en un tableau d'arguments (même format que `argv`). La fonction retourne un tableau contenant tous ces tableaux d'arguments et terminé par la valeur `NULL`.

Par exemple, si la ligne de commande est

```
iutsh$ ls -l | more
```

la fonction retourne le tableau



- La fonction stocke dans le paramètre `nb` le nombre de commandes lues sur la ligne de commandes. Dans le cadre de ce TP, nous ne traiterons pas les tubes, il y aura donc une seule commande par ligne de commandes. Ce paramètre ne sera donc pas utile dans ce TP. Il n'en sera pas de même dans les TP suivants.
- La fonction stocke dans le paramètre `flag` certaines informations concernant la ligne de commandes lues. La valeur stockée est :
  - 1 si la ligne de commandes est valide et est terminée par `&`;
  - 0 si la ligne de commandes est valide mais ne se termine pas par `&`;
  - -1 si une erreur est survenue. Dans ce cas, la fonction retourne la partie de la ligne de commande qu'elle a réussi à construire.
- La fonction

```
void affiche(char *** t);
```

permet d'afficher le résultat de l'appel de la fonction `ligne_commande`. Le but de cette fonction est de contrôler la lecture.

- La fonction

```
void libere(char *** t);
```

désalloue la mémoire utilisée par le tableau passé en paramètre.

## Derniers conseils

**Attention :** un shell est un programme susceptible de fonctionner durant un très long moment, il est donc possible que des erreurs dues à des modifications d'environnement, à une saturation de la mémoire, ...surviennent.

En conséquence n'oubliez pas :

- de libérer (`free`) toutes les zones mémoires que vous allouez ;
- de gérer tous les codes de retour des appels systèmes (signalant éventuellement des erreurs).