IFT2245 | *Travail pratique #1*

Mihai Ionescu - **1012378** André Langlais - **20044886**

February 3, 2019

1 Compréhension de la donnée

1.1 Survol & Cadeaux & Détails

Les sections **ci-haut** étaient très bien expliquées (et détaillées). Nous n'avons donc rencontré aucun problème.

1.2 CH: un shell

Nous avons dû relire quelques fois cette section, car nous ne sommes pas familier avec plusieurs de ces commandes. En particulier les points **2** et **4**.

Nous avons dû faire un peu de recherche (tutoriels et videos éducatives) pour en apprendre davantage sur le fonctionnement de certaines commandes.

2 Compréhension du code

Nous ne sommes pas des experts en **C** (loin de là, ...très loin), mais le code fourni était plutôt simple à comprendre.

Donc, rien à signaler pour cette section!

3 Complétion du code

Alors là! Nous avons pas mal de chose à dire.

Nous avons divisé cette section en plusieurs sous-sections pertinentes qui expliquent les différentes situations que nous avons rencontrées.

3.1 Exécution d'une commande

Pour des raisons que nous ignorons toujours, nous avons constaté qu'il faut utiliser la fonction *execvp* pour executer une commande, et ce, en lui passant **le**

nom de la commande comme premier argument et comme premier élément du second argument (tableau d'arguments).

3.2 Énoncés conditionnels

Lors du **parsing** des commandes et des arguments, nous avons constaté un problème au niveau des énoncés conditionnels : *l'imbrication (illimité) de ceux-ci.*

Notre solution a été de traiter ces cas-là en premier. Donc en gros, nous évaluons (dans l'ordre) les expressions suivantes :

- Énoncés conditionnels.
- Les && et ||.
- Les & et >.
- Les commandes.

3.3 Procédure du découpage des expressions

Pour découper adéquatement les expressions à partir de la chaîne de caractères fournie à notre programme, nous utilisons une approche récursive. À chaque appel, nous effectuons un ou plusieurs forks (dans le cas des && ou ||) pour extraire les différentes expressions.

Nous utilisons **strtok** pour sectionner une chaîne de caractères en plusieurs mots et nous procédons ensuite avec de petits algorithmes pour extraire chaque section de l'expression courante.

Nos algorithmes ont certainement des failles, mais nous avons mis en place quelques vérifications pour assurer que notre programme détecte les entrées invalides. Cela étant dit, il se peut que certains cas ne soient pas couverts (cela entraînera des comportements inconnus).

3.4 Gestion de la mémoire

Comme il s'agit de notre premier programme en C, nous avons eu beaucoup de difficulté avec la gestion de la mémoire. Nous avons essayé de créer des tableaux dynamiques (de tailles différentes) pour sauvegarder les différentes chaînes de caractères. Par exemple, l'entrée suivante :

```
If true ; then ls -la ; done
```

. . . est découpée récursivement en plusieurs tableaux.

Par exemple, nous aurions ici :

```
If true ; then ls -la ; done -> ["true", "ls -la"];
```

Puis, avec la recursion :

```
ls -la ->  ["ls", "-la"]
```

C'est bien beau tout ça, mais nous avons eu beaucoup de difficulté à libérer la mémoire récursivement. Nous avons tenté de gérer la mémoire allouée en sauvegardant l'adresse de chaque pointeur dans des tableaux que nous avons ensuite passé en paramètre récursivement.

Le plus difficile dans tout cela a été de déboguer le code ...et nous n'avons malheureusement pas réussi à intégrer correctement l'allocation dynamique de la mémoire.

Suite à quelques essais, nous avons finalement opté pour une solution simple : *allouer un tableau de taille fixe au début de notre programme.*

Ce tableau contient un maximum de **64 tokens** et est utilisé à travers tout notre programme.

4 Conclusion

Notre plus grande surprise a été de constater le manque d'**IDE**s disponibles pour l'écriture de programme en **C** et la degré de difficulté relié au déboguage.

Outre cela, l'idée de ce **TP** était plutôt intéressante et au final, nous avons approfondi notre compréhension des processus.

THE END