TP 1

Découverte du langage C

1 L'environnement de travail

A. Linux

Histoire [1] *Linux* ou *GNU/Linux* est une famille de systèmes d'exploitation *open source* de type *Unix* fondé sur le *noyau Linux*, créé en 1991 par Linus Torvalds.
En 1991, l'étudiant finlandais Linus Torvalds, indisposé par la faible disponibilité du serveur informatique UNIX de l'université d'Helsinki, entreprend le développement d'un noyau de système d'exploitation, qui prendra le nom de « noyau Linux ».
Linus Torvalds utilisait et appréciait le noyau Minix. Le 25 août 1991, il annonce sur le forum Usenet `comp.os.minix` le développement du noyau Linux [2].
Linus Torvalds choisit rapidement de publier son noyau sous licence GNU GPL. Cette décision rend compatibles juridiquement les systèmes GNU et Linux. Dès lors, pour combler le vide causé par le développement inachevé de Hurd, GNU et le noyau Linux sont associés pour former un nouveau système d'exploitation (parfois considéré comme variante de GNU) : GNU/Linux ou Linux.

Ce que vous avez devant les yeux

- Une interface graphique classique, vous pouvez cliquer sur les icones pour lancer les logiciels qui vont s'ouvrir dans des fenêtres, sur la petite croix rouge en haut à droite de la fenêtre pour la fermer ;
- si vous lancez l'explorateur de fichiers, vous retrouver une organisation arborescente classique pour les fichiers, avec des dossiers, des sous-dossiers… Votre répertoire personnel sous Linux s'appelle communément le « home » ; le chemin complet de ce répertoire est traditionnellement `/home/nom_utilisateur/`.

1. je cite ici Wikipedia
2. I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones.

B. La ligne de commande

Exercice 1.1 Lancez un terminal. Il y a plusieurs moyens de faire cela, suivant la configuration exacte de l'ordinateur :

- chercher dans le menu d'applications ;
- utiliser le raccourci clavier `Ctrl-Alt-T` s'il est configuré ;
- cliquer sur le raccourci présent dans la barre de lancement rapide ;
- appuyer sur la touche *Windows* du clavier, puis commencer à taper « terminal » jusqu'à ce que la bonne application soit sélectionnée…

Vous obtenez une *invite de commande*, dans laquelle vous allez pouvoir interagir avec un *shell* (ou *interpréteur de commandes*). Fondamentalement, ce shell joue le même rôle qu'un shell Python : il permet d'exécuter des instructions écrites dans un langage de programmation (*bash*, en l'occurrence) et de voir s'afficher le résultat de ces instructions. Il y a cependant une grosse différence avec la boucle interactive d'un langage de programmation : le shell va surtout servir à exécuter des programmes pré-existants, écrits dans un langage quelconque au départ (mais mis sous préalablement sous une forme exécutable pour la machine).

Exercice 1.2 **Quelques commandes au hasard.** Exécutez les commandes suivantes :

1. `whoami` (qui suis-je ?)
2. `date` (ça devrait être clair)
3. `htop` (utilisation des ressources ; appuyez sur `F1` pour avoir de l'aide, `F10` ou `q` pour quitter)
4. `python` (pour quitter, exécuter `exit()` ou appuyer sur `Ctrl-D`)
5. `ping google.com` (pour quitter appuyer sur `Ctrl-C`, ce qui est une règle générale quand une commande ne termine pas)

Exercice 1.3

1. Exécuter la commande `gedit hello.py`. Cela lance l'éditeur de texte `gedit` et ouvre le fichier `hello.py` (qui est créé à cette occasion, puisqu'il n'existait pas).
2. Dans l'éditeur, taper la ligne `print("Hello, world!")` puis sauvegarder le fichier (`Ctrl-S`) et quitter (`Ctrl-Q`).
3. Vous revenez dans le terminal : exécuter à présent la commande `python hello.py`.

La dynamique du CTRL-C

Dans les combinaisons de touche, du type « `CTRL-C` » ou « `ALT-F4` », les deux touches n'ont PAS des rôles symétriques. À 18 ans, il n'est pas trop tard pour comprendre ça (10 ans plus tard, ce sera plus difficile). Vous attribuez à « doigt 1 » et « doigt 2 » deux doigts d'une même personne. Au ralenti, ça donne :

- Doigt 1 appuie sur la touche `CTRL`, et reste appuyé dessus jusqu'à nouvel ordre. Doigt 2 ne fait rien.
- Après un long moment, doigt 2 va finalement taper brièvement mais fermement sur la touche `C`. Doigt 1 est toujours sur `CTRL`, puisqu'on ne lui a pas dit de bouger.
- Bien plus tard, doigt 1 libère enfin la touche `CTRL`.

S'il vous faut dix secondes pour accomplir correctement ce mouvement, ce n'est pas un problème. S'il vous faut un dixième de seconde pour le faire n'importe comment, vous aurez régulièrement des ennuis et ne saurez jamais utiliser un clavier décemment. Normalement très rapidement, il vous faudra une demi-seconde (et probablement moins) pour faire le BON mouvement ; bravo.

Quelques champs d'application du même principe :

- `CTRL-C` : pour copier un fichier, ou un morceau de texte préalablement sélectionné (grisé) sous un éditeur.
- `CTRL-X` : pour couper (un fichier ou un morceau de texte préalablement sélectionné). Il est mis dans le « presse-papiers », donc peut être ensuite collé.
- `CTRL-V` : pour coller (un fichier ou un morceau de texte) qui aura préalablement été mis de côté via un *copier* ou un *couper*.
- `CTRL-Z` : pour annuler la dernière action effectuée.
- `CTRL-S` : pour sauver le fichier sur lequel on est en train de travailler (sur quasiment tous les éditeurs, dont Word, Libreoffice, VS Code…)

Enfin, il convient de révéler à certains l'utilisation de quelques touches mystérieuses du clavier. Remontons depuis la touche `CTRL` à gauche du clavier.

- `SHIFT` : sert à faire des majuscules... ou bien obtenir un chiffre sur la clavier principal.
- `CAPS LOCK` : ne sert PAS à obtenir une majuscule, contrairement à ce que pensent certains. Il est utile pour faire n majuscules, avec $n \geqslant 10$. En deçà de dix, on peut laisser un doigt appuyé sur `SHIFT`. Ici encore, 18 ans est un âge où on peut encore changer des habitudes absurdes [3].
- La touche dessus `CAPS LOCK` est la touche de tabulation, qui permet d'écrire grosso modo un bloc d'espace ; et surtout à activer la complétion automatique.
- La touche bizarre avec un 2 : elle n'existe pas, en fait ! Oubliez-la (et NON, on ne l'utilisera pas pour calculer le carré de quelque chose).
- La touche `ALT GR` à droite de la touche d'espace est celle qui permet d'obtenir les symboles en bas à droite des touches de chiffres : par exemple, on obtient les crochets via `ALT GR-(` et `ALT GR-)`.

Obtenir de l'aide

Exercice 1.4

1. Exécuter la commande `ls` (pour *list* : cette commande liste les fichiers du répertoire courant).

En réalité, cette commande admet un très grand de variantes, gouvernées par des *flags*. Si l'on souhaite par exemple un affichage détaillé avec un fichier par ligne, trié par taille décroissante, on pourra exécuter une commande du type `ls -X -Y` où `X` et `Y` sont bien choisis. Comme personne ne connaît toutes les options de toutes les commandes, plusieurs mécanismes sont prévus pour obtenir de l'aide.

2. Exécuter la commande `ls -help`. Et puis exécuter `date -help`, `ping -help`…
3. Quelle commande faut-il exécuter pour que `ping` s'arrête au bout de 3 réponses ?

Pour `ls`, l'aide obtenue avec `-help` est un peu indigeste. Il y a des moyens d'en extraire l'information que l'on recherche, mais il est souvent plus pratique d'utiliser les *man pages* (les pages du manuel). Pour cela on tapera `man nom_de_la_commande`. Les man pages sont en général très complètes.

4. Exécuter la commande `man ls` et essayer de trouver les options à utiliser pour obtenir l'affichage évoqué plus haut (un fichier par ligne, avec détails, trié par taille décroissante).
5. `gcc` est le compilateur C le plus couramment utilisé sous Linux. Exécuter `man gcc`, et constater que parfois, il peut y avoir un peu trop d'informations…

3. Il semblerait que l'utilisation systématique de `CAPS LOCK` pour faire UNE majuscule ne soit pas si rare… C'est probablement à cause d'une mauvaise compréhension de la dynamique du `SHIFT` : « Ben quand j'essaie d'appuyer sur les deux touches en même temps, souvent ça ne marche pas… ». Ben oui, forcément…

Système de fichiers

Pour se déplacer et se repérer dans l'arborescence, les commandes de base sont :

- `pwd` pour *Print Working Directory*, qui affiche le chemin complet du répertoire courant ;
- `cd` pour *Change Directory* pour se déplacer dans un nouveau répertoire. On peut utiliser un chemin absolu (`cd /etc`) ou relatif, qui commence implicitement dans le répertoire courant. Tout répertoire contient deux liens spéciaux : « `.` » qui pointe vers le répertoire lui-même et « `..` » qui pointe vers le répertoire parent. `cd .` équivaut donc à rester dans le répertoire courant, et `cd ..` permet de remonter dans le répertoire parent ;
- `ls` que nous avons déjà vu pour lister les fichiers dans un répertoire. Sans argument, `ls` liste le contenu du répertoire actuel. On peut aussi donner à `ls` un chemin absolu commençant à la racine du système de fichiers (par exemple `ls /etc/ssh`) ou un chemin relatif.

Pour gérer les fichiers, les commandes de base sont :

- `mkdir nom_du_repertoire` pour créer un répertoire portant le nom `nom_du_repertoire` dans le répertoire courant ;
- `touch nom_du_fichier` pour créer un fichier vide ;
- `rm nom_du_fichier` pour effacer un fichier (attention, l'action est irréversible) ;
- `cp` et `mv` pour copier et déplacer des fichiers.

Exercice 1.5

1. Dans votre répertoire personnel, créer un sous-répertoire `informatique_tp`.
2. Dans ce nouveau répertoire, créer un sous-répertoire `TP1`.
3. Dans ce nouveau répertoire, créer un fichier vide `exercice06.c`.

II Programmer en C

Vous pouvez maintenant lancer *VS Code*. Ce logiciel (édité par *Microsoft*) est un éditeur moderne qui offre tout le confort attendu aujourd'hui pour écrire du code :

- coloration syntaxique (des mots-clés, des parenthèses…) ;
- complétion automatique ;
- affichage des erreurs…

Si vous avez vos habitudes avec un autre éditeur de texte (*emacs*, *vim*, *sublime text*…), vous pouvez bien sûr l'utiliser !

A. Entrées-sorties

On rappelle que les entrées-sorties textuelles dans la console sont possibles grâce à la bibliothèque `stdio.h` (pour *standard input-output header*).

1. Sortie dans le terminal

On utilise pour cela la fonction `printf`.

- Si on veut afficher une chaîne de caractères constante, il suffit de la donner comme argument à la fonction :

```
printf("mp2i\n");
```

  Le `\n` final permet de rajouter un retour à la ligne ;

- si l'affichage souhaité dépend de la valeur d'une variable, on indiquera dans la chaîne où placer cette valeur avec les instructions de formatage suivantes :
  - `%d` pour un entier,
  - `%f` pour un flottant,
  - `%c` pour un caractère…

  et on donnera comme second argument à la fonction `printf` l'expression que l'on veut afficher :

```
int k = 2;
printf("mp%di\n", k);
```

- on peut pousser plus loin le principe précédent en ajoutant plusieurs instructions de formatage, il suffira alors de donner autant d'arguments supplémentaires à la fonction d'affichage en respectant l'ordre d'apparition dans la chaîne :

```
int j = 42;
printf("Le carré de %d vaut %d, son cube vaut %d et sa racine %f\n",
       j, j * j, j * j * j, sqrt(j));
```

Exercice 1.6

1. Remplir le fichier `exercice06.c` pour écrire un programme qui écrit dans le terminal « Bienvenue en mp2i! ».
2. Observer les messages d'erreur à la compilation :
   (a) si l'on enlève la ligne `#include <stdio.h>` (ce sera sans doute seulement un *warning*);
   (b) si l'on enlève le point-virgule après le `return 0` (cette fois, ce sera une erreur).

   Noter aussi que ces problèmes sont signalés en direct sous VS Code.
3. Revenir à une version sans erreur du fichier, la compiler, puis exécuter les deux commandes ci-dessous dans le terminal :

```
$ ./hello
$ echo $?
```

   La deuxième commande demande l'affichage (`echo`) du contenu de la variable spéciale `?`. Cette variable contient le code de retour de la dernière commande exécutée, et donc `0` ici puisque notre fonction main renvoie `0`.
4. Remplacer le `return 0;` par `return 1;`, recommencer les étapes ci-dessus et vérifier que le code de retour a bien changé.

2. Entrée dans le terminal

On utilise pour cela la fonction `scanf`, et on va retrouver les instructions de formatage précédentes. Par exemple, pour lire la valeur d'un entier et stocker cette valeur dans une variable `i`, on écrira :

```
int i; /* il faut obligatoirement que i soit déclaré */
scanf("%d", &i);
```

Remarquez l'esperluette & devant le nom de la variable dans la fonction `scanf`; celle-ci est indispensable ici, on expliquera prochainement son rôle.
Par ailleurs, la variable peut déjà contenir une valeur avant le `scanf`, cette dernière sera écrasée par la valeur lue.

B. Exercices

Exercice 1.7

1. Écrire une fonction `int absolue(int x)` qui renvoie la valeur absolue de son argument. *Cette fonction existe déjà, et s'appelle* `abs`*. Il faut inclure l'entête* `stdlib.h` *pour y avoir accès.*
2. Écrire une fonction `void affiche_si_pair(int x)` qui affiche l'entier x s'il est pair. L'opérateur modulo s'écrit `%` en C.
3. Tester ces deux fonctions en écrivant un petit programme, en le compilant et en l'exécutant.

Exercice 1.8 Écrire un programme qui (après compilation et exécution) va demander la valeur d'une borne supérieure et produire l'affichage suivant :

```
$ ./exercice08
Quelle est la borne supérieure ? 8
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
```

Exercice 1.9 Nous calculons la racine carrée d'un nombre supposé positif (sans vérifier pour l'instant ce point), par une stratégie algorithmique classique, la *dichotomie*. Le principe de la dichotomie est le suivant :

- on cherche la réponse dans un intervalle,
- si la réponse est au milieu de l'intervalle, on a trouvé la solution,
- sinon
  - soit elle est inférieure à ce milieu et on regarde dans la moitié de gauche en suivant la même méthode,
  - soit elle est supérieure et on regarde dans la moitié de droite en suivant la même méthode.

Ici, si on note x le nombre dont on cherche la racine carrée, on sait que celle-ci appartient à l'intervalle $[1, x]$ si $x \geqslant 1$ et $[x, 1]$ sinon. On peut alors comparer le carré de la valeur candidate (ici $\frac{x+1}{2}$) à x : s'ils sont suffisamment proches, on considère qu'on a trouvé $\sqrt{x}$, sinon on continue dans la bonne moitié de l'intervalle.

Écrire un programme qui demande à l'utilisateur d'entrer un flottant et qui affiche sa racine avec une précision de 10^{-3}.

Exercice 1.10 On définit la série harmonique comme :

$$\forall n \in \mathbb{N}^*,\ H_n = \sum_{k=1}^{n} \frac{1}{k}.$$

On démontre (très facilement) que cette suite est croissante et (assez facilement) qu'elle tend vers $+\infty$. On définit alors, pour tout x réel :

$$f(x) = \min\{n \in \mathbb{N}^* \mid H_n \geqslant x\}.$$

Écrire un programme donnant l'affichage suivant :

```
$ ./exercice09
Quelle est la borne supérieure ? 4
f(0.000000) = 0
f(0.500000) = 1
f(1.000000) = 1
f(1.500000) = 2
f(2.000000) = 4
f(2.500000) = 7
f(3.000000) = 11
f(3.500000) = 19
f(4.000000) = 31
```

On essayera de faire en sorte que le programme soit capable d'afficher le résultat jusqu'à $f(15)$ en temps raisonnable.

Exercice 1.11 On veut afficher des cibles de la forme suivante (ici avec 4 lettres) :

```
aaaaaaa
abbbbba
abcccba
abcdcba
abcccba
abbbbba
aaaaaaa
```

Écrire un programme qui demande le nombre de lettres voulues à l'utilisateur et qui affiche la cible correspondante.
Pour comprendre comment gérer facilement les différents caractères pour cet exercice, vous pouvez tester le petit code suivant :

```c
#include <stdio.h>

int main(void) {
  char lettre = 'a';
  int code_lettre = (int)lettre;  // on transforme le caractère en entier

  for (int i = 0; i < 5; i++) {
    // dans le «printf» suivant, on travaille sur l'entier,
    // avant de le retransformer en caractère
    printf("%c\n", (char)(code_lettre + i));
  }

  return 0;
}
```

Exercice 1.12 Écrire une fonction `bool est_premier(int n)` qui renvoie `true` ou `false` suivant si n est premier ou non (on pourra supposer $n \geqslant 1$). Attention, pour disposer du type `bool` avec les constantes `true` et `false`, il faudra ajouter la ligne suivante au début du fichier :

```c
#include <stdbool.h>
```

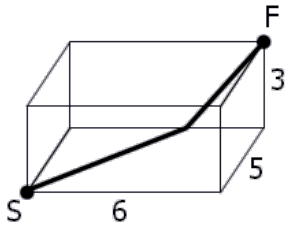
Tester cette fonction en écrivant un programme donnant l'affichage suivant :

```
$ ./exercice11
Quelle est la borne supérieure ? 11
1 n'est pas premier
2 est premier
3 est premier
4 n'est pas premier
5 est premier
6 n'est pas premier
7 est premier
8 n'est pas premier
9 n'est pas premier
10 n'est pas premier
11 est premier
```

Pour occuper les plus rapides

Exercice 1.13 **Project Euler – Problem 86**

A spider, *S*, sits in one corner of a cuboid room, measuring 6 by 5 by 3, and a fly, *F*, sits in the opposite corner. By travelling on the surfaces of the room the shortest "straight line" distance from *S* to *F* is 10 and the path is shown on the diagram.



However, there are up to three "shortest" path candidates for any given cuboid and the shortest route doesn't always have integer length. It can be shown that there are exactly 2060 distinct cuboids, ignoring rotations, with integer dimensions, up to a maximum size of *M* by *M* by *M*, for which the shortest route has integer length when $M = 100$. This is the least value of *M* for which the number of solutions exceeds two thousand; the number of solutions when $M = 99$ is 1975.
Find the least value of M such that the number of solutions exceeds one million.

Exercice 1.14 **Project Euler – Problem 297**

Each new term in the Fibonacci sequence is generated by adding the previous two terms.
Starting with 1 and 2, the first terms will be : 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. Every positive integer can be uniquely written as a sum of nonconsecutive terms of the Fibonacci sequence. For example, $100 = 3 + 8 + 89$. Such a sum is called the Zeckendorf representation of the number.
For any integer $n > 0$, let $z(n)$ be the number of terms in the Zeckendorf representation of n. Thus, $z(5) = 1,\ z(14) = 2,\ z(100) = 3\ldots$
Also,

$$\sum_{n=1}^{10^6-1} z(n) = 7\,894\,453.$$

Find

$$\sum_{n=1}^{10^{17}-1} z(n).$$