TP 1
Révisions

Durée prévue : 1 séance (1h30)

Environnement de TP Pour les TPs il est **obligatoire** d'utiliser un système « de type Linux [1] », et de faire le maximum de manipulations sur la ligne de commande. Pour ce TP, vous devez avoir le compilateur C++ de GCC installé (dans une version un peu récente) : pour mémoire, il s'agit de la commande `g++` que vous avez déjà utilisée dans d'autres UE. Il est aussi **fortement recommandé** de vous mettre à un éditeur de texte plus adapté que `gedit` pour programmer, mais nous vous laissons déterminer lequel vous convient le mieux (mais il faut au moins qu'il indente, numérote les lignes et qu'il colore).

1.1 La ligne de commande

EXERCICE 1 ▶ **Le minimum vital**

Juste quelques questions pour réviser des commandes shell utiles. On aura l'occasion de revoir plus de choses en cours de semestre, mais on se concentre pour l'instant sur le strict nécessaire :

- `mkdir` de l'anglais *make directory*,
- `rmdir` pour *remove directory*,
- `ls` pour *list on screen*,
- `cd` pour *change directory*,
- `cp` pour *copy*,
- `mv` pour *move*,
- `grep` (acronyme de *global regular expression print*!),
- `cat` (pas évident, mais ça vient de *concatenate*...),
- `less` (successeur de `more`...),
- `apropos` et `man`.

Commencez en ouvrant un terminal de commandes. Ensuite :

1) Créez un répertoire de travail `LIFASR5` avec `mkdir`.

.....

2) Vérifiez que le répertoire a bien été créé avec `ls`.

.....

3) Faites de `LIFASR5` votre répertoire courant avec `cd`.

.....

4) Que fait la commande `pwd` ? De quoi est-elle l'acronyme en anglais ?

.....

5) Créez un sous-répertoire `TP1`.

.....

6) Entrez la commande `cd` (sans paramètre) : que devient votre répertoire courant ? (donnez le plus de réponses possible, au moins trois !)

.....

7) En utilisant `cd` et la complétion automatique avec la touche « tabulation », retournez dans le répertoire `~/LIFASR5/TP1`.

.....

8) Avec `cp` (*copy*), copiez le fichier `/etc/passwd` dans le répertoire courant.

.....

1. (Unix plus précisément ; cela peut être : une distribution GNU/Linux, WSL sous Windows 10, MacOSX, ou une distibution GNU/Linux dans une machine virtuelle).

9) Avec `cat`, affichez le contenu du fichier sur la sortie standard.

.....

10) Même chose avec `less` (touche vers le haut et vers le bas pour faire défiler le texte, `q` pour quitter).

.....

11) Renommez votre copie de `passwd` en `mdp` avec `mv`.

.....

12) Affichez toutes les lignes du fichier `mdp` qui contiennent la chaîne `nologin` avec `grep`.

.....

13) Déplacez le fichier `mdp` dans le répertoire `/tmp/`.

.....

14) Supprimez le fichier `/tmp/mdp` avec `rm`.

.....

15) Que retourne la fonction C `getenv`? Répondez en consultant la page de manuel de cette fonction (`man getenv`, puis touche `q` pour quitter).

.....

16) Cherchez de l'aide sur la fonction C `write`, qui permet d'écrire sur un descripteur de fichier (tentez `man write`, puis utilisez `apropos write` pour identifier la bonne page).

.....

1.2 Compilation de petits projets

EXERCICE 2 ► **Compilation et Makefile**

Pour les TP qui viennent, il est important de savoir compiler et exécuter des programmes depuis la ligne de commande : sans cela, vous ne pourrez pas travailler sur la plupart des sujets. Le but de cet exercice est de rappelez comment utiliser simplement un compilateur C++ standard, `g++` pour fixer les idées (mais vous pourriez aussi utiliser `clang++`), et comment créer un `Makefile` simple pour compiler de petits projets.

1ère partie, compilation en ligne de commande :

1) Commencez par créer un fichier source `test.cpp`, qui contient un programme quelconque, par exemple :

```
#include <iostream>
using namespace std;
int main(void) {
  cout << "Hello!" << endl;
  return 0;
}
```

Compilez ce programme avec `g++ test.cpp` : quel fichier a été créé (`ls`)? Comment voyez-vous qu'il s'agit d'un fichier exécutable (`ls -l`)? Comment faire pour l'exécuter?

.....

.....

2) Supprimer le fichier exécutable généré à la question précédente (avec `rm`), re-compiler avec `g++ -o toto test.cpp`, puis faites un `ls -l` : que permet de faire l'option `-o`?

.....

3) D'abord, supprimez le fichier généré à la question précédente avec `rm toto`. Quelle commande utiliser pour que le source `test.cpp` soit compilé en l'exécutable `test`?

.....

4) Supprimer le fichier généré à la question précédente. Dans la catégorie des classiques du rire, on va ajouter une fonction dans un fichier séparé. Déclarez la fonction suivante dans un fichier `fonction.cpp` :

```
double cube(double x) {
  return x*x*x;
}
```

Créez le fichier d'entête `fonction.h`, qui contient le *prototype* de la fonction `cube`. Dans le `main` de `test.cpp`, ajoutez la ligne

```
cout << "Le cube de 33.0 est " << cube(33.0) << endl;
```

Tentez de générer un exécutable à partir de `test.cpp` : de quoi se plaint le compilateur?

.....

5) Ajouter la ligne `#include "fonction.h"` au début du fichier `test.cpp`, puis tentez à nouveau de compiler : de quoi se plaint maintenant le compilateur? Expliquez.

.....

.....

6) On peut se débrouiller en créant un fichier objet à partir de `fonction.cpp` : utilisez la commande `g++ -c fonction.cpp`, et vérifiez qu'un fichier `fonction.o` a bien été créé. Pour mieux comprendre [2] :

- vérifiez que `fonction.o` ne contient pas le code C de la fonction cube (`cat fonction.o`).
- testez la commande `nm fonction.o` (`man nm` pour regarder la signification de `T` dans le manuel).
- désassembler le fichier objet avec `objdump -d fonction.o` : le code en langage machine, et le code en langage d'assemblage de votre fonction doivent s'afficher! Jouons « à Champollion » : quelle instruction est utilisée pour les deux multiplications effectuées dans la fonction `cube`?

.....

.....

.....

Compilez maintenant `test.cpp` avec `g++ -o test test.cpp fonction.o` et vérifiez que vous obtenez bien l'exécutable `test`. Expliquez pourquoi, enfin, tout se passe bien!

.....

.....

7) Avec `rm *.o test`, supprimez les fichiers de la question précédente. On va essayer maintenant une approche plus directe, avec la commande :
`g++ -o test test.cpp fonction.cpp`
A nouveau, vérifiez que tout se passe bien, et expliquez. Quel peut être l'inconvénient de cette méthode?

.....

2ème partie, utilisation simplifiée d'un `Makefile` : Supprimez les fichiers `test` et `fonction.o` de la partie précédente. Un `Makefile` est un fichier qui contient des recettes (règles) pour fabriquer de nouveaux fichiers à partir de fichiers sources. Essentiellement, une recette est de la forme suivante :

```
fichier à faire: liste des fichiers nécessaires
_____méthode pour fabriquer le fichier d'après les fichiers nécessaires
```

Insistons sur l'importance d'insérer le caractère de tabulation avant la méthode de fabrication (sinon, le `Makefile` plante quand on l'invoque avec `make`). Par exemple, voici une règle pour fabriquer un fichier objet :

```
fonction.o: fonction.cpp fonction.h
_____g++ -c fonction.cpp #ceci est un commentaire
```

Ici, on se propose d'écrire un `Makefile` tout simple pour le programme de la partie précédente.

1) Commencez par écrire un `Makefile` de deux lignes qui contient une seule recette pour fabriquer l'exécutable `test` de la partie précédente (approche « directe »).

.....

.....

Exécutez votre recette en entrant que la ligne de commande `make test` : vérifiez que le programme est bien compilé. Entrez à nouveau `make test` : que constatez-vous? Modifiez le fichier `fonction.h`, en ajoutant un commentaire sur ce que fait la fonction `cube`, puis entrez encore `make test` : que constatez-vous?

.....

2) En général, on ajoute (au moins) deux règles spéciales au `Makefile` :

2. Il faudra peut-être que vous intalliez de nouveaux packages dans votre distribution ; passer en cas de problème.

— `all: <liste des fichiers à fabriquer>`. Comme ça, on peut se contenter d'entrer `make` au lieu de faire suivre la commande des noms des fichiers à fabriquer.

— une règle pour faire le nettoyage, en supprimant les exécutables, les `.o` et autres fichiers temporaires. Cela peut être :

```
clean:
        rm -f *.o *~ test
```

Ajoutez une recette `all` et une recette `clean` à votre `Makefile`. Notez que lorsqu'un `Makefile` contient plusieurs recettes, chaque règle est séparée de ses voisines par une ligne vide. Vérifiez que tout fonctionne bien.

3) Ajoutez la ligne suivante dans le `main` de `test.cpp` :

```
cout << "La racine cubique du cube de 33.0 est " << cbrt(cube(33.0)) << endl;
```

Tentez ensuite de compiler. Logiquement, il doit y avoir un problème : le prototype de la fonction `cbrt` n'est pas connu à la compilation. En vous aidant du `man cbrt` (touche `q` pour quitter le `man`), expliquez ce problème, et corrigez-le !

.....

Utilisation d'une bibliothèque : Le petit Nicolas vient d'écrire programme `mdp.c` pour essayer de comprendre le fonctionnement de la fonction `char *crypt(const char *key, const char *salt)` :

```
#include <iostream>
#include <crypt.h>

int main(void) {
  char clair[] = "Toto2001FaitDuVelo";
  char *hache = crypt(clair, "66");
  std::cout << "clair : " << clair << std::endl;
  std::cout << "hache : " << hache << std::endl;
}
```

Mais toutes les tentatives de compilation du petit Nicolas avec `g++ -o mdp mdp.c -Wall` se terminent par l'erreur

```
/tmp/cc7GKCQ1.o : Dans la fonction "main"
mdp.c:(.text+0c4c) : référence indéfinie vers "crypt"
collect2: error: ld returned 1 exit status
```

Il vous est demandé de secourir le petit Nicolas en répondant aux questions suivantes :

1) De quel type d'erreur s'agit-il ?

.....

2) Quelle est la bibliothèque manquante ?

.....

3) Corrigez la ligne de compilation pour qu'elle fonctionne et essayez.

.....

4) Sur votre système, où se trouve la bibliothèque manquante ?

.....

5) Même si ça n'était pas le but premier de l'exercice : que fait le programme du petit Nicolas ?

.....

EXERCICE 3 ▶ Programmation en C/C++

Cet exercice ne porte par directement sur la programmation système, mais son but est de réviser la programmation en C/C++, un peu les pointeurs, et d'apprendre à se documenter avec les pages du manuel.

Vous devez écrire un programme dans lequel vous fabriquez une chaîne de caractères de type `string` (`std::string` pour être précis), qui comporte deux lignes :

— sur la première ligne, vous mettrez la chaîne `"Date du jour : "` suivi de la date du jour au format "jj/mm/aaaa",

— sur la deuxième ligne, vous mettrez `"Heure courante : "` suivi de l'heure courante (sur 24h, avec les heures et les minutes séparée par la lettre h).

Pour cela, vous devez utiliser les fonctions C suivantes :

— `strftime()` (`man strftime`),

— `localtime()` (`localtime`),

— `time()` (`man 2 time`).

Vous utiliserez aussi les opérateurs "+" et "+=" pour concaténer des chaînes de la classe `string`. Évidemment, vous devez écrire un Makefile et testez votre programme. Voici un exemple produit par l'affichage de la chaîne :

```
Date du jour : 22/01/2021
Heure courante : 11h09
```