# Travaux Dirigés et Pratiques de Systèmes d'Exploitation (HLIN303)

Michel Meynard

## Préambule

Les exercices de TD et de TP sont destinés à vous aider à comprendre le cours. En TP, vous devrez avoir recours aux pages du manuel Unix/Linux disponibles par la commande `man xxx`, ou par l'URL `http://manpagesfr.free.fr/consulter.html`. Certaines distributions utilisent aussi la commande `info` pour décrire les commandes complexes.

## 1 Introduction, source, objet, compilation

**Exercice 1 (TD)** Quelle différence existe-t-il entre un interpréteur et un compilateur ? Citez deux exemples pour chacun d'entre eux.

**Exercice 2 (TD)** Quelles similitudes et quelles différences existe-t-il entre un fichier d'en-tête (toto.h) et une bibliothèque ? Quand les utilise-t-on dans le processus de compilation ? Citez deux exemples pour chacun d'entre eux.

**Exercice 3 (TD)** Quelle différence entre l'inclusion d'un fichier d'entête standard tel que `stdio.h, stdlib.h, ctype.h` et un fichier d'entête personnel `monprog.h` ?

**Exercice 4 (TD)** Comment les paramètres de la ligne de commande sont-ils passés à un programme C ? Comment l'environnement du processus (variables d'environnement telles que PATH, HOME, ...) est-il atteignable par un programme C ?

**Exercice 5 (TD/TP)** On souhaite écrire un programme affichant le nombre de paramètres passés à la ligne de commande, ces paramètres, ainsi que les variables d'environnement.

1. Ecrire l'algorithme de ce programme.
2. Ecrire le programme C correspondant.

**Exercice 6 (TP)** A l'aide du programme précédent, testez les possibilités du compilateur `gcc` en produisant :
- le fichier prétraité ;
- le fichier assembleur ;
- le fichier objet ;

Observez ces différents fichiers.

**Exercice 7 (TD/TP)** On souhaite réaliser le calcul de la moyenne d'une suite de 5 nombres flottants saisis au clavier.

1. Ecrire l'algorithme de ce programme.
2. Ecrire le programme C correspondant.

**Exercice 8 (TD/TP)** En utilisant la fonction `atoi` qui convertit une chaîne en entier, réaliser le calcul de la moyenne de la suite des paramètres passés à la ligne de commande : `moy 5 8 12 3`.

1. Ecrire l'algorithme de ce programme.
2. Ecrire le programme C correspondant.

**Exercice 9 (TD/TP strsplit)** Dans certains fichiers structurés, les articles sont représentés sur une ligne dont les champs sont séparés par un caractère séparateur (fichiers .csv). On souhaite écrire une fonction qui découpe une chaîne de caractères correspondant à une ligne csv en un tableau de chaînes dynmiques. Par exemple, l'appel `strsplit("/bin:/usr/bin:/usr/local/bin",':')` doit générer le tableau suivant :

```
0 --> /bin
1 --> /usr/bin
2 --> /usr/local/bin
3 NULL
```

1. Ecrire l'algorithme de ce programme.
2. Ecrire le programme C correspondant.

## 2 Types de données, compilation séparée

**Exercice 10 (TD/TP)** Soit une chaîne de caractères C contenant un nombre entier positif en représentation décimale. On veut écrire une fonction convertissant cette chaîne en un entier.

1. Ecrire un algorithme itératif de conversion.
2. Ecrire la fonction C correspondante.

**Exercice 11 (TD/TP)** On veut réaliser la conversion inverse de l'exercice précédent. Soit un nombre entier positif que l'on veut convertir en une chaîne de caractères C contenant sa représentation décimale.

1. Ecrire un algorithme de conversion.
2. Ecrire la fonction C correspondante.

**Exercice 12 (TD)** Ces algorithmes de conversion (entier vers chaîne, chaîne vers flottant, ...) sont-ils fréquemment employés ? Précisez à quelles occasions.

**Exercice 13 (TD/TP)** Soit la **définition** des fonctions suivantes :

— fonction pair

```
int pair(unsigned int i){
  if (i==0)
    return 1;
  else
    return impair(i-1);
}
```

— fonction impair

```
int impair(unsigned int i){
  if (i==0)
    return 0;
  else
    return pair(i-1);
}
```

1. Créer un fichier `pair.c` (resp. `impair.c`) qui contienne la définition de la fonction pair (resp. impair). Créer un fichier `pair.h` (resp. `impair.h`) qui contienne la **déclaration** (prototype) de la fonction pair (resp. impair). Créer un programme principal `spair.c` qui réalise l'algorithme suivant :

   ```
   lire l'entier positif n passé à la ligne de commande
   si pair(n) alors
     afficher : n est pair !
   sinon
     afficher : n est impair !
   ```

   Bien entendu, il faut réfléchir au différentes inclusions à réaliser dans les différents fichiers puis compiler les 3 sources .c et lier avec le nom spair et enfin tester ce programme.
2. Ecrire un fichier ppair.c contenant les trois fonctions pair(), impair() et main(). Compiler et lier cet unique fichier en un exécutable ppair. Tester ppair.

## 3 Code de Huffman

Le codage de Huffman (1952) est une technique de compression de données permettant de stocker un message (un fichier) ou de le transmettre grâce à un minimum de bits afin d'améliorer les performances. Ce codage à taille variable suppose l'indépendance des caractères du message et on doit connaître la distribution probabiliste de ceux-ci à une position quelconque dans ce message. Plus la probabilité d'occurrence d'un caractère dans un fichier est grande, plus son code doit avoir une taille réduite.

On code chaque caractère par un mot de longueur variable de façon à ce qu'aucun mot ne soit le préfixe d'un autre. La propriété principale des codes de Huffman consiste en ce que la longueur moyenne du codage d'un caractère dans un fichier soit minimale.

Pour calculer le code d'un alphabet et de sa distribution, on construit un arbre binaire étiqueté par des 0 et des 1, les feuilles de cet arbre représentant les caractères tandis que les chemins issus de la racine constituent les codes correspondants.

**Exemple 1**

*Soit l'alphabet a,b,c,d,e associé à la distribution probabiliste de la table 1.*

*On obtient l'arbre binaire de Huffman de la figure 1.*

*En parcourant cet arbre de la figure 1 on aboutit au code de Huffman correspondant à la table 2.*

*Remarquons que ce code n'est pas unique (il suffit de permuter les 0 et les 1). On calcule la longueur moyenne de codage comme suit :*

| Caractère | a | b | c | d | e |
|---|---|---|---|---|---|
| Probabilité | 0,3 | 0,25 | 0,20 | 0,15 | 0,10 |

TABLE 1 – Distribution probabiliste

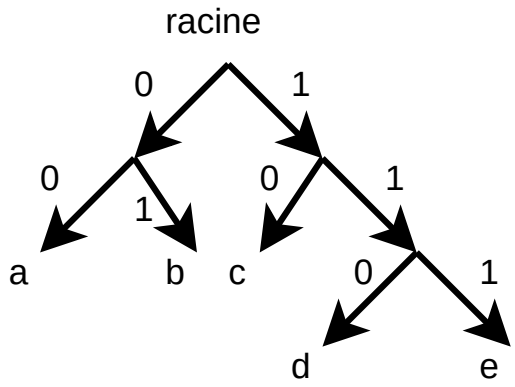


FIGURE 1 – Arbre de Huffman

— 75% *des caractères nécessitent 2 bits (a,b,c) ;*
— 25% *des caractères nécessitent 3 bits (d,e) ;*
— $L_{moy} = 75\% * 2 + 25\% * 3 = 2,25$ *bits*

*Remarquons qu'avec un code de taille fixe, chaque caractère aurait nécessité 3 bits, soit une perte de* 33%.

### 3.1 Algorithme de Huffman

L'algorithme consiste à construire l'arbre en partant des deux feuilles "les plus basses" (plus faibles probabilités ici d et e). On leur associe un père, sorte de noeud virtuel dont la probabilité d'apparition du préfixe associé est égale à la somme des probabilités de ses deux fils. On réitère le processus en choisissant à nouveau les deux sommets sans père de plus faible probabilité jusqu'à la construction de la racine.

**Exercice 14 (TD)** A l'aide de l'exemple précédent, construisez l'arbre de Huffman en précisant à chaque pas les noeuds créés.

**Exercice 15 (TD/TP)** Soit la distribution probabiliste de la table 3 pour les dix caractères représentant les chiffres décimaux.

1. Etablir manuellement un arbre de Huffman pour cette distribution en expliquant votre démarche.
2. En utilisant le résultat précédent, établir le code de Huffman pour cette distribution de la table 3 en expliquant votre démarche.
3. Donner la longueur moyenne de codage d'un chiffre du message.

**Exercice 16 (TD/TP)** On souhaite calculer la distribution probabiliste des caractères d'un fichier passé en paramètre où les caractères sont stockés sur 8 bits.

1. Décrire une structure de données à base de tableau permettant de calculer puis de stocker la distribution probabiliste.
2. Ecrire l'algorithme pour calculer la distribution probabiliste.
3. Ecrire le programme C correspondant.

**Exercice 17 (TD/TP)**

1. Décrire une structure de données à base de tableau permettant de construire l'arbre de Huffman.
2. Ecrire l'algorithme pour construire l'arbre ;
3. Ecrire l'algorithme pour calculer le code de Huffman ; Huffman associé.
4. On souhaite implémenter cet algorithme sur des fichiers où les caractères sont stockés sur 8 bits. Quelle est la taille minimale et maximale d'un code ? Comment stocker le tableau des codes dans le fichier codé ?
5. (Projet) Programmer l'algorithme de compression et de décompression en C.

| Caractère | a | b | c | d | e |
|---|---|---|---|---|---|
| Code | 00 | 01 | 10 | 110 | 111 |

TABLE 2 – Code de Huffman

| Caractère | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Probabilité | 0,05 | 0,1 | 0,11 | 0,11 | 0,15 | 0,06 | 0,08 | 0,2 | 0,07 | 0,07 |

TABLE 3 – Distribution probabiliste

# 4 Compilation séparée et bibliothèque

**Exercice 18 (TD)** Listez les avantages et inconvénients de la liaison statique et dynamique.

**Exercice 19 (TP)** En reprenant l'exercice 13, on souhaite construire une bibliothèque statique (`.a`) contenant les fonctions `pair` et `impair` à l'aide de la commande `ar`.

1. Créer les fichiers objets `pair.o` et `impair.o`. Puis éditer leurs liens avec le fichier `spair.c` après compilation en construisant l'exécutable `spair2`.
2. Créer une bibliothèque statique `libpair.a` contenant les deux fichiers objets `pair.o` et `impair.o`. Vérifier le contenu de `libpair.a`.
3. Compiler spair.c et éditer ses liens avec votre bibliothèque dans un exécutable spair3. Tester spair3.

**Exercice 20 (TP)** En reprenant l'exercice 13, on souhaite construire une bibliothèque dynamique (`.so`) contenant les fonctions `pair` et `impair` à l'aide de la commande .

1. Créer la bibliothèque dynamique `libpair.so` contennat les fichiers objets `pair.o` et `impair.o`.
2. Créer une bibliothèque statique `libpair.a` contenant les deux fichiers objets `pair.o` et `impair.o`. Vérifier le contenu de `libpair.a`.
3. Compiler `spair.c` dans un exécutable `spair4` puis tester ses liens dynamiques.

**Exercice 21 (TP bibliothèque sd)** Vous trouverez sur le site `http://www.lirmm.fr/~meynard/` des fichiers d'entête et des sources C définisssant des conteneurs de types génériques :

— liste générique : listeg
— association ou dictionnaire : assog
— ensemble : ensg
— arbre binaire ! arbin g
— types de base (entier, chaine, flottant, car) : types_base

L'objectif du TP est de construire une bibliothèque statique et une bibliothèque générique contenant les objets définis, puis de l'utiliser en écrivant une application simple : calcul de toutes les parties d'un ensemble fini d'entier.

# 5 Entrées/Sorties

**Exercice 22 (TD)** Quelle différence entre l'utilisation d'appels systèmes `man 2` (open, read, close) ou bien des fonctions de bibliothèque `man 3` (fopen, fprint, ...) ?

**Exercice 23 (TD/TP)** On souhaite compter les caractères d'un fichier passé en argument à la ligne de commande en utilisant des appels systèmes (`wc -c`).

1. Ecrire un algorithme
2. Ecrire le programme C correspondant.

**Exercice 24 (TD/TP)** On souhaite refaire l'exercice précédent en utilisant des fonctions de bibliothèque. Que faut-il faire ? Faites-le.

**Exercice 25 (TD/TP)** Un filtre est un programme qui lit des données sur l'entrée standard et qui rejette dans sa sortie standard les données modifiées. On souhaite écrire un filtre `monhead` qui ne rejette que les n premières lignes d'un fichier. Par exemple :

```
head -3 monfic.txt
```

Cette commande permet d'afficher les 3 premières lignes du fichier `monfic.txt`.

1. Qu'est-ce qu'une ligne ? Faut-il utiliser appel système ou fonction de bibliothèque ?
2. Ecrire un algorithme
3. Ecrire le programme C correspondant.
4. l'option `-c43` de `head` permet de lire les 43 premiers caractères. Implémenter cette option.

# 6 Filtres et Entrées/Sorties formattées

**Exercice 26 (TD/TP)** On souhaite écrire un filtre `montail` qui ne rejette que les n dernières lignes d'un fichier.

1. Quelle structure de données nécessite ce programme ? Décrivez précisément cette structure de données !
2. Ecrire un algorithme
3. Ecrire le programme C correspondant.
4. l'option `-c43` de `tail` permet de lire les 43 derniers caractères. Implémenter cette option.

**Exercice 27 (TD)** Etudiez le programme suivant : **programme testaffichint.c**

```
#include <stdio.h>/* printf */
int main(int argc, char *argv[], char *env[]) {
  int i=12345;
  printf("%4.0d\n",i);
  fwrite(&i,sizeof(int),1,stdout);
}
```

L'affichage obtenu est le suivant :

```
$testaffichint
12345
90  $
```

1. Quelle différence esiste-t-il entre ces deux affichages ?
2. Précisez la valeur de `90__`. Quel est le "boutisme" (endianness) de cette machine ?
3. Quels avantages et inconvénients de ces types de codage dans les fichiers en matière de sauvegarde de données ?

**Exercice 28 (TD/TP fichiers .csv)** En examinant certains fichiers de configuration situés dans le répertoire `/etc`, tels que `passwd`, `fstab`, `group`, on voit que ces fichiers textes sont composés d'articles d'une ligne, chaque article étant composé de champs séparés par un caractère (`:`). On souhaite écrire une commande `monselect` correspondant de manière simplifiée à l'instruction `SELECT` de SQL. Les champs étant numérotés de 1 à n, l'exemple suivant affichera chaque ligne du fichier `passwd` dont le deuxième champ est égal à "dupont" :

```
monselect /etc/passwd : 2 dupont
```

1. Ecrire un algorithme
2. Ecrire le programme C correspondant.

**Exercice 29 (TP sauvegarde binaire)** Soit un tableau d'étudiants que l'on veut sauver (écrire) dans un fichier au format binaire. Suivent les définitions de type et de ce tableau :

```
typedef struct {
  int code;     /* code étudiant 20091234 */
  char nom[20]; /* "dupont" */
  char dnais[8]; /* 19901231 */
} etudiant;
etudiant tabetu[100]={{20091234,"dupont","19891231"},
              {20091235,"martin","19891130"},
              {0,"",""}
}; /* tableau des étudiants */
```

Le tableau est compacté vers les indices faibles et le premier article (struct)ayant un code nul indique la fin des étudiants.

1. Ecrire quelques instructions C permettant d'afficher le tableau des étudiants.
2. Ecrire un programme C permettant de sauver le tableau dans un fichier binaire passé en argument et ne contenant que les étudiants présents.
3. Visualiser le fichier sauvé en hexadécimal (`hexl` ou `od -cx`) et donner une interprétation.
4. Ecrire un programme C permettant de lire le fichier sauvé afin de le stocker dans un tableau puis afficher le résultat.