IUT — DUT 1re Année
Université de Marne la Vallée — Année 2012-2013

Prog UNIX : Entrées / sorties et fichiers d'en-tête
TP n°2

Dans tous les exercices, on veillera particulièrement à la lisibilité de code. Le but de ce TP est la mise au point d'un répertoire de personnes. On mémorise un ensemble de personnes, et un ensemble d'informations sur chacune de ces personnes.

Ces informations sont :

- le nom,
- le prénom,
- le genre (masculin, féminin),
- la date de naissance,
- les groupes (ami, université, travail, ...)

Quitte à modifier sa fonction `main` extrêmement souvent, il est très vivement conseillé de tester les fonctions une par une, juste après leur implantation. Par souci d'efficacité, toutes les variables de type structuré seront passées par adresses aux fonctions (que ces dernières soient modifiées ou pas par ces fonctions).

Exercice 1 **Manipulation de date**

Dans un fichier `repertoire.h`, définissez la structure suivante:

```
1 typedef struct date{
2   int jour;
3   int mois;
4   int annee;
5 }Date;
```

Une date est ainsi définie par trois numéros : celui du jour, du mois et de l'année. Écrire dans un fichier `repertoire.c` les fonctions suivantes manipulant des variables de type `Date`.

- Écrire une fonction `void affiche_date(Date *d)` qui affiche la date donnée en argument dans le terminal.
- Écrire une fonction `void saisir_date(Date *d)` permettant de saisir une date au clavier. On vérifiera que la valeur du champ jour soit comprise entre 1 et 31, la valeur du champ mois soit comprise entre 1 et 12 et enfin que l'année soit un entier positif plus petit que 2013.
- Écrire une fonction `int meme_mois(Date *d1, Date *d2)` qui retourne 1 si les deux dates sont le même mois (l'année pouvant être différente) et qui retourne 0 sinon.

Exercice 2 **Manipulation de fiche**

Dans le fichier `repertoire.h`, rajoutez les deux macros et la structure suivantes:

```
#define MAXCHAINE 50
#define MAXGROUPE 10

typedef struct fiche{
  char nom[MAXCHAINE];
  char prenom[MAXCHAINE];
  char genre;
  Date naissance;
  int groupe[MAXGROUPE];
}Fiche;
```

Ainsi une fiche décrira une personne via son `nom` (limité à 50 caractères), son `prénom` (limité à 50 caractères), un caractère pour le genre, une date de naissance et un tableau de 10 entiers (à voir comme un tableau de 10 booléens VRAI/FAUX). La personne appartient au groupe lorsque l'entrée associée dans le tableau vaut 1, la personne n'appartient pas au groupe lorsque cette même entrée vaut 0. Rajoutez dans le fichier `repertoire.c` les fonctions suivantes manipulant des variables de type `Fiche`.

- Écrire une fonction `void affiche_fiche(Fiche *f)` qui affiche la fiche donnée en argument dans le terminal.
- Écrire une fonction `void saisir_fiche(Fiche *f)` permettant de saisir une fiche au clavier. Il est vivement conseillé d'utiliser ce qui a été fait dans l'exercice 1.
- Écrire une fonction `int nombre_groupe(Fiche *f)` qui retourne le nombre de groupes dans lesquels la personne associée à la fiche appartient.

Exercice 3 **Manipulation de répertoire**

Un répertoire est un ensemble borné de fiches. Dans le fichier `repertoire.h`, rajoutez la macro et la structure suivantes:

```
#define MAXFICHE 50

typedef struct repertoire{
  Fiche pers[MAXFICHE];
  int taille;
}Repertoire;
```

Le répertoire est un tableau partiellement rempli qui contient au plus 50 fiches significatives. La taille de la partie renseignée du répertoire est en fait stockée dans le champ taille de la structure.

- Écrire une fonction `void affiche_repertoire(Repertoire *R)` qui affiche les fiches du répertoire dans la console. Il est très vivement conseillé d'utiliser les fonctions déjà implantées.

- Écrire une fonction `void sauve_repertoire(Repertoire *R)` qui sauvegarde dans un fichier `repertoire.txt` les fiches contenues dans le répertoire passé en argument. Une solution pour en faciliter la lecture après sauvegarde consiste à fabriquer un fichier où chaque fiche occupe une ligne du fichier, et chaque ligne contient les informations bien ordonnées d'une fiche. Voici un exemple possible pour deux fiches :

  ```
  Dupont Albert (h) 31/01/1954 1 1 0 1 0 0 1 0 0 0
  Faure Pierre (h) 24/07/1976 0 1 0 1 0 1 0 0 0 0
  ```

- Écrire une fonction `void charge_repertoire(Repertoire *R)` qui lit le fichier `repertoire.txt` et rempli le répertoire passé en argument avec les informations contenues dans le fichier. Ce chargement de répertoire est facilité si les informations du fichier sont bien ordonnées.

- Écrire une fonction `int ajoute_fiche(Repertoire *R, Fiche *f)` qui ajoute la fiche passée en argument dans le répertoire passé en argument.

Il convient maintenant de créer un répertoire `data` dans votre TP. La place adaptée pour le fichier `repertoire.txt` est le dossier `data` (attention aux chemins apparaissant dans votre code si vous déplacez des fichiers...). Vous pouvez maintenant commencer la finalisation, toute idée améliorant la qualité du rendu est bienvenue.

Exercice 4 **Finalisation du rendu**

Voici le cahier des charges pour ce second TP :

- votre travail devra être rendu en pièce attachée d'un mail envoyé à une adresse communiquée par votre chargé de TP,

- vous ne devrez envoyer qu'une seule pièce jointe au format `tar.gz` dont le nom devra être : `TP1_NOM_PRENOM.tar.gz` où `NOM` est votre nom de famille et `PRENOM` votre prénom (voir l'aide mémoire `tar` en ligne),

- le sujet de votre mail devra être : `[DUT1 info][TP2 prog_sys] NOM PRENOM` où `NOM` devra être remplacé par votre nom de famille et `PRENOM` par votre prénom,

- votre rendu devra contenir au moins trois répertoires aux noms de `doc` ,`sources`, `data` et un fichier nommé `Makefile`,

- une fois votre TP décompressé, la commande `make` devra compiler les sources C des exercices situées dans le dossier `sources` et fabriquer un exécutable au nom adapté qui se placera dans la racine de votre TP. La compilation devra être effectuée avec les options `-Wall` et `-ansi`,

- L'exécutable de votre rendu devra tester, autant que possible, toutes les fonctions implantées dans vos sources.

- la commande `make clean` devra effacer tous les exécutables générés lors de l'exécution de la commande `make`.