TP8 - Sérialisation

Langage C (LC4)

Semaine du 17 mars 2014

1 Serialisation

Le but de ce TP est de manier des fichiers, et d'apprendre une méthode de sauvegarde de données, en utilisant un format de TLV (type, length, value). Un tel format permet d'encoder des données binaires, de manière modulaire et extensible.
Afin d'avoir quelque chose à sauvegarder, nous nous donnons la structure de carnet d'adresses suivante :

```
struct personne {
    int id; /* On prendra l'indice du tableau, par exemple */
    char *nom;
    char age;
    char telephone[10];
    char *notes;
};
```

Nous définissons ici les différents TLVs. Tout TLV commence par deux champs : le type (sur 8 bits), et la taille du TLV (sur 16 bits). La taille du TLV n'inclus pas les 3 octets du type et de la taille elle-même. Par exemple, le champ taille du TLV de la section 2.1 vaut 9. Une implémentation qui ne connaîtrait pas le type d'un TLV peut l'ignorer et, grâce à sa taille, passer au TLV suivant.

Question 1. Définissez un tableau statique de `struct personne` contenant un certain nombre d'entrées choisies aléatoirement, ainsi qu'une variable statique donnant le nombre d'entrées dans le fichier.

Question 2. Écrivez une fonction `int serialize(void)` qui ouvre un fichier `sauvegarde.dat` en écriture avec la fonction `fopen(3)` (faites la gestion d'erreur !), puis le ferme avec `fclose(3)` [1], et quitte. Nous construirons la fonction au fur et à mesure ; elle renverra 0 en cas de réussite, et -1 en cas d'échec.

Note Dans la suite, nous écrirons les données octets par octets, avec `fputc`, en mettant les octets de poids fort d'abord (grand-boutiste, ou *big-endian*). On fera les opérations à la main, en utilisant décalages et masques.

Question 3. Écrivez une fonction `int put_16(FILE *stream, short s)` qui écrit `s` dans `stream` (en grand-boutiste). La fonction renvoie 0 en cas de réussite, -1 en cas d'échec. On utilisera les opérations bit-à-bits : décalages et masques.

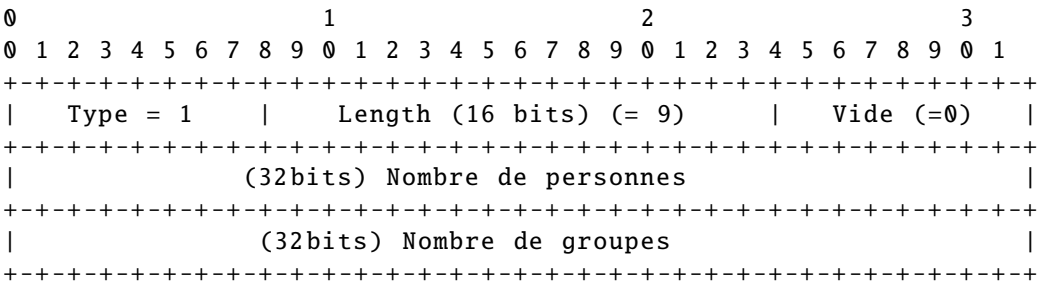
Question 4. Écrivez sur le même modèle `int put_32(FILE *stream, int i)`.

1. Fermer les fichiers ouverts est aussi important que de libérer les zones de mémoires allouées dynamiquement, car des ressources sont allouées pour chaque fichier ouvert.

2 Carnet de base, écriture

2.1 Métadonnées

Nous allons commencer par sauvegarder des métadonnées dans notre fichier, et nous nous fixons comme norme de commencer notre fichier par un TLV de métadonnées. Celui-ci contient le nombre de personnes du carnet d'adresses, et le nombre de groupes (nous verrons plus tard, mettons 0 pour le moment) :



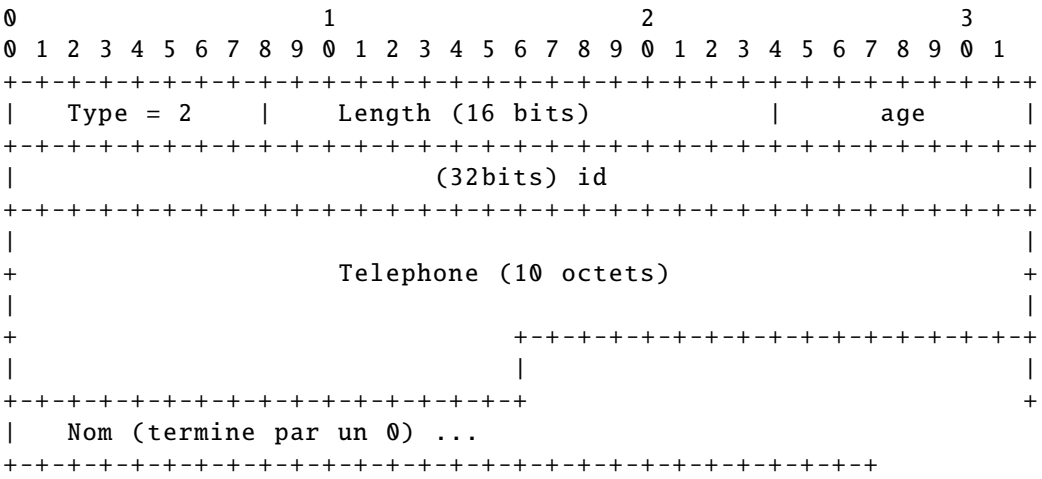
Note Le champ `Vide` ne sert que visuellement (c'est plus joli), et pourrait être utilisé pour des extensions futures. Il sera ignoré.

Question 5. Modifiez `serialize` pour qu'elle enregistre un TLV de métadonnées dans le fichier de sauvegarde.

Question 6. On pourra vérifier que le fichier produit a bien les octets voulus en regardant son contenu binaire, avec `hexdump -C <file>`.

2.2 Entrées du carnet

Les entrées du carnet sont définies comme suit :



Question 7. Modifiez `serialize` pour qu'elle enregistre toutes les entrées du carnet sous forme de succession de TLVs d'entrées. Ces TLVs seront après le TLV de métadonnées. (Oui, les notes sont omises, c'est voulu.)

3 Carnet de base, lecture

Il est temps de voir si on est capable de lire le fichier en mémoire !

Question 8. Écrivez une fonction `int get_16(FILE *stream, short *s)` qui lit 2 octets depuis `stream`, et stocke leur interprétation grand-boutiste dans `s`.

Question 9. Faites de même avec `int get_32(FILE *stream, int *i)`.

Question 10. Écrivez une fonction `int jette(FILE *stream, int noctets)` qui lit (et ignore) `noctets`.

Question 11. Créez deux variables statiques `struct personne *carnet_lu` et `int taille_carnet_lu`.

Question 12. Écrivez une fonction `int deserialize(void)` qui ouvre un fichier `sauvegarde.dat` en lecture seule, puis qui le ferme.

Question 13. Modifiez `deserialize` pour qu'elle soit capable de lire un TLV de métadonnées. Elle modifiera aussi `carnet_lu` en allouant la mémoire nécessaire à contenir tous les TLV du carnet d'adresses.

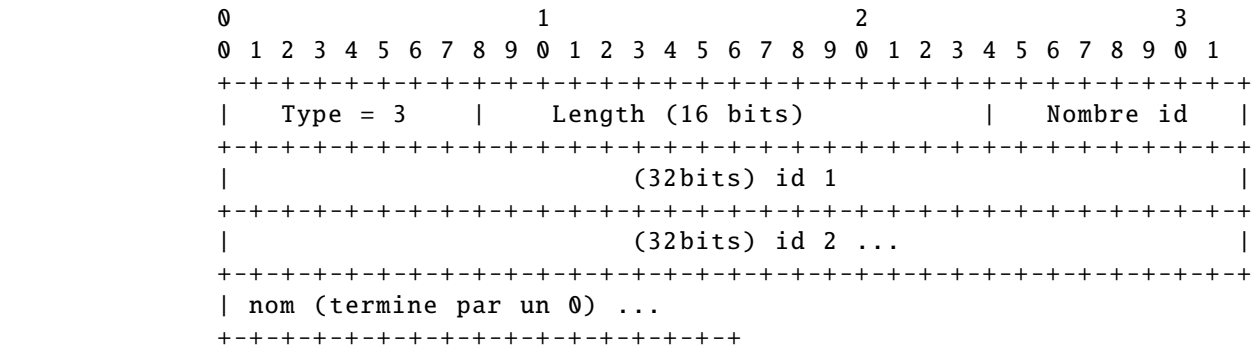
Question 14. Modifiez `deserialize` pour qu'elle lise les TLV d'entrées de carnet, et les stocke dans `carnet_lu`.

Question 15. Comparez votre carnet avant et après enregistrement. J'espère que ce sont les mêmes ! !

4 Carnet étendu

4.1 Groupes

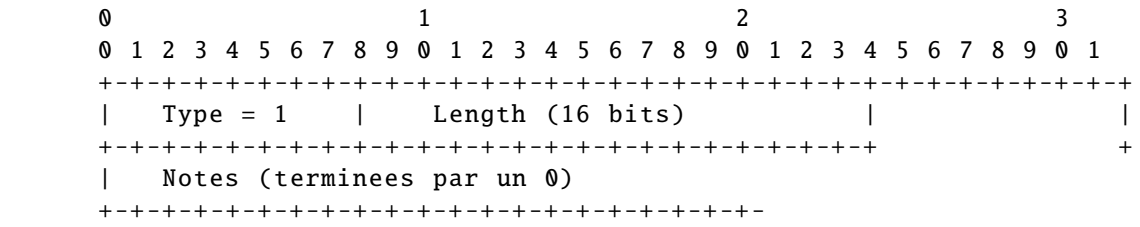
On désire maintenant faire de groupes de contacts. Un groupe de contact est constitué d'une liste d'identifiant de contacts (255 au maximum), et d'un nom de groupe. En voici la spécification :



Question 16. Implémentez la lecture et l'écriture d'un groupe, et créez les données (statiques) nécessaires.

4.2 Sous-TLVs, et principes d'extensions

Nous avons un carnet d'adresse qui fonctionne, cependant nous voulons pouvoir l'enrichir de certaines extensions facultatives, comme par exemple les notes, qui sont présentes dans notre structure de données. Pour cela, deux mécanismes sont possibles : utiliser de nouveaux TLVs, ou utiliser des sous-TLVs. Nous allons utiliser le 2e procédé.
Puisque la taille d'un TLV est définie, dans le 2e champ, rien ne nous empêche d'ajouter des informations au bout : nous allons ajouter un sous-TLV au TLV 2.



Question 17. Implémentez le mécanisme de sous-TLV aussi bien en écriture qu'en lecture.