TP 5 : Conversion de fichiers

Notions : Structures, fichiers textes et fichiers binaires

1 Fichier texte : bibliothèque de chansons

A partir du fichier texte contenant une bibliothèque de titre de chansons, vous allez construire un fichier binaire qui sera exploité par un programme permettant de visualiser, modifier, ajouter, supprimer des informations dans le fichier binaire.

Les informations que vous devrez gérer sont le titre de la chanson, l'interprète, le format (AIF, MP3, etc..), un identifiant (un entier qui permettra de retrouver cette chanson) et une note mise par les internautes.

1.1 Le problème posé, les données et le résultat attendu

Dans le fichier texte, les informations sont classées par ordre alphabétique des interprètes.

On trouve donc successivement les informations suivantes : l'identifiant, le format, le nom de la chanson, l'interprète, la note, pour chaque chanson, comme ci dessous (figure 1).

id	Format	Titre	Interprète	Note
3212	MP3	Take On Me	a ha	0.9
6365	WAV	Mysterious Ways	Angélique Kidjo	2
2408	MP3	Rolling in the Deep	ADELE	5.1
4142	AIF	Lovesong	ADELE	1.8

FIGURE 1 – Extrait du fichier texte.

Dans le fichier binaire que vous allez créer, ces informations seront classées par ordre d'identifiant, qui fera office de clé. La position des informations sur une chanson dans le fichier est donc donnée par son identifiant : c'est la clé primaire ou moyen d'accès direct à cette chanson. Les fiches contenant les informations sont stockées les unes derrière les autres, sans aucun séparateur. Le format d'une chanson est une chaîne de 8 caractères, l'interprète ou auteur, ainsi que le titre, sont des chaînes de 80 caractères, l'identifiant est un entier, la note est un réel double précision. Le fichier binaire aura alors l'allure de la figure 2.

Chaque fiche correspondant à une chanson occupe la même taille dans le fichier binaire et comporte les informations dans le même ordre (voir figure 2). Donc, le début de l'élément de rang n est à la position $(n)\times$taille d'une fiche (la première fiche est vide et n'est pas utilisée). Pour lire ou écrire une fiche dans le fichier binaire, après avoir ouvert le fichier, il faut donc se positionner au bon endroit dans le fichier (fonction `fseek`) et lire (fonction `fread`) ou écrire (fonction `fwrite`) le nombre d'octets correspondants à une fiche.

Fiche 1 : Strawberry Fields Forever position en octets : 184	1 int 12 car 1 double 80 car 80 car	1 WAV 5.7 Strawberry Fields Forever The Beatles
Fiche 2 : (I Can't Get No) Satisfaction position en octets : 368	1 int 12 car 1 double 80 car 80 car	1 MP3 3.5 (I Can't Get No) Satisfaction The Rolling Stones
Fiche 3, position 552	...	...

FIGURE 2 – Extrait du fichier binaire.

La structure ci dessous (figure 3) représente une chanson. Cette structure est utilisée par le programme que nous vous fournissons pour gérer la table. La taille peut être obtenue avec l'opérateur `sizeof` par `sizeof(CHANSON)`

```
#define MAX_CHAINE 80
#define MAX_FORMAT 12
typedef struct chanson {
  int id;
  char format[MAX_FORMAT];
  double note;
  char titre[MAX_CHAINE];
  char auteur[MAX_CHAINE]; };
typedef struct chanson CHANSON;
```

FIGURE 3 – Structure de données représentant les informations.

1.2 Travail à réaliser

Télécharger et vérifier que vous pouvez bien visualiser le fichier texte `mesmusiques.txt` à l'aide d'un éditeur de texte ou de la commande unix `cat mesmusiques.txt` .

Vous devez ensuite écrire un programme qui lit le fichier texte `mesmusiques.txt` , ligne par ligne et réécrit ces informations sur les chansons dans un fichier binaire ordonné selon les `id`. Chaque ligne contient toutes les informations sur une chanson, que l'on doit mettre dans une variable structurée de type CHANSON. Le programme effectuera donc les actions suivantes, tant qu'il y aura des chansons à lire :

1. la lecture d'une fiche ou chanson se fera facilement à l'aide de la fonction `fscanf`,
2. vous calculerez l'endroit où doit se trouver cette fiche dans le fichier binaire grâce à son identifiant,
3. vous positionnerez le descripteur de fichier binaire au bon endroit (fonction `fseek`, voir les pages du manuel par `man fseek`),
4. vous écrirez en une seule instruction toute la structure de l'élément dans le fichier binaire (fonction `fwrite`).

Remarquez que vous lisez les fiches dans l'ordre d'apparition du fichier texte, mais que vous les écrivez à la bonne place dans le fichier binaire, dans lequel elles sont ordonnées par identifiant et non par ordre alphabétique. Ainsi, la première fiche lue sera celle de `Take On Me`, qui devra être écrite en $3212^{\text{ième}}$ position (identifiant de cette chanson), soit à l'octet `3212*sizeof(CHANSON)` du fichier binaire `mesmusiques.bin`.

Remarques :

1. Vous n'utiliserez pas de tableau de structure. Une seule structure sera utilisée pour contenir les informations sur une chanson, soit une ligne du fichier texte.

2. Une structure complète ne peut pas être lue directement dans un fichier texte. Il faut lire chaque champ de la structure, l'un après l'autre, avec la fonction `fscanf`. Cette fonction retourne le nombre d'informations qui ont été correctement lues dans le fichier. L'instruction ci-dessous retourne la valeur 5 si toutes les informations sont lues correctement :

```
fscanf(source,"%d %[^\t] %[^\t] %[^\t] %lf ",&(ch.id), ch.format,ch.titre,ch.auteur,&(ch.note));
```

   Cela correspond à la lecture dans le fichier de descripteur `source` des informations suivantes :

   (a) ch.id : lecture d'un entier,

   (b) ch.format : lecture d'une chaîne jusqu'à trouver une tabulation,

   (c) ch.titre : lecture d'une chaîne jusqu'à trouver une tabulation,

   (d) ch.auteur : lecture d'une chaîne jusqu'à trouver une tabulation,

   (e) ch.note : lecture d'un réel double précision.

3. Le programme `test_chanson` dans le répertoire `/users/prog1a/C/librairie/bin` permet de lire un fichier binaire correctement formé.

4. la commande `od -t x1 mesmusiques.bin` permet d'examiner le fichier binaire octet par octet, en hexadécimal. Vous devez trouver à peu près la sortie figure 4. La première colonne est le numéro de l'octet en base 8 (20 en octal correspond à 16 en décimal). Les autres valeurs sont les octets du fichier, regroupés par ligne de 16 valeurs pour la visibilité. Les 4 premiers `00 00 00 00` sont l'`id` de la chanson. Les 12 octets suivants sont le format `41 49 46 00 00 00 00 00 00 00 00 00` en hexadécimal, soit `A I F \0 \0 \0 \0 \0 \0 \0 \0 \0` en ASCII. Les 8 octets suivants sont le réel double précision avec le codage IEEE 754 illisible `9a 99 99 99 99 99 f1 3f` en hexadécimal, soit `232 231 231 231 231 231 361 ?` en ASCII. Puis les 80 suivants sont le titre `41 6d 65 72 69 63 61 6e` ou `A m e r i c a n` ..... Les 80 octets pour l'auteur commencent en 0000150 en octal (soit 104 en décimal) et valent `44 6f 6e 20 4d 63 4c 65` ... ou `D o n M c L e` ..... On a ainsi un enregistrement complet des informations de la première chanson. On arrive alors à l'octet 0000270 (en octal, soit 184 en décimal), où l'on trouve l'identifiant suivant, c'est à dire 1 sur 4 octets, etc ...

```
0000000    00  00  00  00  41  49  46  00  00  00  00  00  00  00  00  00
0000020    9a  99  99  99  99  99  f1  3f  41  6d  65  72  69  63  61  6e
0000040    20  50  69  65  00  00  00  00  00  00  00  00  00  00  00  00
0000060    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
*
0000140    00  00  00  00  00  00  00  00  44  6f  6e  20  4d  63  4c  65
0000160    61  6e  00  00  00  00  00  00  00  00  00  00  00  00  00  00
0000200    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
*
0000260    00  00  00  00  00  00  00  00  01  00  00  00  57  41  56  00
0000300    00  00  00  00  00  00  00  00  cd  cc  cc  cc  cc  cc  16  40
0000320    53  74  72  61  77  62  65  72  72  79  20  46  69  65  6c  64
0000340    73  20  46  6f  72  65  76  65  72  00  00  00  00  00  00  00
0000360    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
*
0000440    54  68  65  20  42  65  61  74  6c  65  73  00  00  00  00  00
0000460    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
*
0000560    02  00  00  00  4d  50  33  00  00  00  00  00  00  00  00  00
```

FIGURE 4 – Premières lignes de la commande `od -tx1 mesmusiques.bin | more`

5. la commande `od -a mesmusiques.bin` permet d'examiner le fichier binaire octet par octet, en affichage de type caractères. Vous devez trouver à peu près la sortie figure 5.

```
0000000   \0  \0  \0  \0   A   I   F  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000020  232 231 231 231 231 231 361   ?   A   m   e   r   i   c   a   n
0000040        P   i   e  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000060   \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0000140   \0  \0  \0  \0  \0  \0  \0  \0   D   o   n       M   c   L   e
0000160    a   n  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000200   \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0000260   \0  \0  \0  \0  \0  \0  \0  \0 001  \0  \0  \0   W   A   V  \0
0000300   \0  \0  \0  \0  \0  \0  \0  \0 315 314 314 314 314 314 026   @
0000320    S   t   r   a   w   b   e   r   r   y       F   i   e   l   d
0000340    s       F   o   r   e   v   e   r  \0  \0  \0  \0  \0  \0  \0
0000360   \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0000440    T   h   e       B   e   a   t   l   e   s  \0  \0  \0  \0  \0
0000460   \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0000560  002  \0  \0  \0   M   P   3  \0  \0  \0  \0  \0  \0  \0  \0  \0
```

FIGURE 5 – Premières lignes de la commande `od -a mesmusiques.bin | more`

2 Et un fichier d'index : des playlistes

Pour enchaîner des chansons différentes et pouvoir les rejouer, nous allons créer des playlistes et les stocker dans un fichier. Une même chanson pouvant apparaître dans plusieurs playlistes, nous ne souhaitons pas stocker les informations des chansons dans les playlistes : dupliquer ces informations augmenterait la taille du fichier de manière importante [1]. Nous allons utiliser le numéro des chansons dans ces playlistes. Les informations sur les chansons resteront stockées une seule fois, dans le fichier binaire que vous venez de créer.

2.1 Les playlistes

Une playliste est simplement l'ensemble des identifiants des chansons de la playliste. Ces identifiants sont donc des index, permettant de retrouver les informations sur la chanson, dans le fichier des chansons. Lorsque l'on veut afficher les informations sur les chansons de la playliste, il suffit de récupérer les identifiants des chansons, de chercher les chansons correspondant à ces identifiants dans le fichier des chansons puis d'afficher ces informations.

Pour simplifier, une playliste sera limitée à 10 chansons. Elle sera représentée par une structure contenant le tableau des 10 identifiants, le nombre réel de chansons de la playliste (il peut y avoir des playlistes avec moins de 10 chansons) et le numéro de la playliste.

```
#define NB_CH_LIST 10

typedef struct {
  int liste[NB_CH_LIST];
  int id;
  int nb;
} PLAYLIST;
```

2.2 Quel fichier de chansons utiliser ?

Nous pourrions utiliser le fichier texte initial `mesmusiques.txt`. Pour retrouver une chanson dont l'identifiant est `3121` par exemple, il faut alors ouvrir le fichier, lire la première ligne correspondant à la première chanson et tester si son identifiant est 3121. Si oui, nous pouvons afficher les informations.

1. imaginez la taille des fichiers si nous avions de plus le véritable contenu sonore !

Sinon, on passe à la ligne suivante et ainsi de suite. Dans le pire des cas, il faut lire toutes les lignes (plus de 7000 dans ce fichier) [2]. En moyenne, il faut lire la moitié des lignes. Le temps de recherche peut donc devenir très long si le fichier est important.

Si nous utilisons le fichier binaire de chansons, dans lequel les chansons sont rangées par identifiant, chacune occupant le même espace physique, il devient facile de trouver la chanson dont l'identifiant est `3121`. Il faut ouvrir le fichier, positionner le descripteur de fichier à la position `3121*sizeof(CHANSON)` et simplement lire toutes les informations de la chanson. La recherche est donc directe et non séquentielle : elle ne dépend pas du nombre de chansons du fichier.

2.3 Le fichier de playlistes

Les différentes playlistes seront alors écrites dans un fichier texte [3]. Les playlistes seront stockées dans le fichier dans l'ordre de leur entrée par l'utilisateur. Si la playliste numéro 10 est la première que l'utilisateur entre, ce sera la première inscrite dans le fichier. Si l'utilisateur saisit ensuite la playliste numéro 5, ce sera la deuxième playliste dans le fichier et ainsi de suite.

La figure 6 donne un exemple de fichier texte contenant 3 playlistes, avec leur numéro, le nombre de chansons et les numéros des chansons de la playliste. La playliste numéro 5 contient 3 chansons de numéros 100, 200 et 300. La playliste numéro 2 contient 4 chansons de numéros 8, 9, 11 et 12. Chaque ligne du fichier contient une playliste.

```
5 3 100 200 300
2 4 8 9 11 12
8 5 100 1000 2000 3000 4000
```

FIGURE 6 – Un fichier contenant les 3 playlistes numéro 5, 2 et 8.

2.4 La gestion du fichier de playlistes

Les différentes actions autorisées sont les suivantes :

Ajout d'une nouvelle playliste : pour stocker une nouvelle playliste dans le fichier, il faut simplement se positionner en fin de fichier et écrire la nouvelle playliste pointée par `pl` dans le fichier `fpl`.
Le prototype de la fonction sera `void fprintfPlaylist(FILE* fpl, PLAYLIST* pl);`

Afficher la playliste numéro `id` : il faut d'abord lire la playliste, puis afficher les titres des chansons, comme sur la figure 7. Pour lire la playliste de numéro `id`, il faut donc se positionner au début du fichier et lire toutes les playlistes jusqu'à trouver la playliste recherchée ou la fin de fichier. Chaque playliste occupant un espace différent dans le fichier, on ne peut pas se positionner au début d'une playliste en se basant sur la taille occupée dans le fichier. La fonction retourne 1 si la playliste existe, 0 sinon. Son prototype sera `int fscanfPlaylistNumber(FILE* fpl, PLAYLIST* pl, int id) ;`

Suppression d'une playliste : pour supprimer la playliste `id`, on ne peut pas supprimer la playliste directement dans le fichier car sa taille et son emplacement dans ce fichier sont inconnus. Une solution consiste à utiliser un fichier temporaire en procédant de la manière suivante [4] :

1. ouvrir le fichier de playlistes,
2. ouvrir un fichier temporaire, `.playlistes.tmp` [5] par exemple,
3. lire toutes les playlistes dans le fichier initial et les copier dans le fichier temporaire jusqu'à la playliste à supprimer,
4. lire la playliste à supprimer, sans rien en faire,

2. et c'est bien pire dans un cas réel.
3. puisque vous venez de voir la lecture dans un fichier texte et les lecture et écriture dans un fichier binaire, autant voir maintenant l'écriture dans un fichier texte pour des aspects pédagogiques. Un fichier binaire serait bien plus adapté.
4. cette procédure montre que le format texte n'est pas du tout approprié à cette tâche.
5. avec un point en début de nom, ce fichier est un fichier caché. Il n'apparaîtra pas avec la commande `ls`. Il faut utiliser la commande `ls -a` si besoin.

```
Numero de la playlist : 5
Chansons de la playlist :
0 Titre: White Rabbit ; Auteur: Jefferson Airplane  ; format: WAV ; note: 6.8  ; id : 100
1 Titre: Home To You ; Auteur: The Peasall Sisters  ; format: WAV ; note: 4.0  ; id : 200
2 Titre: Lovefool ; Auteur: The Cardigans  ; format: WAV ; note: 6.0  ; id : 300
Fin Playlist  ---------------------
Numero de la playlist : 2
Chansons de la playlist :
0 Titre: Sgt. Pepper Medley ; Auteur: The Beatles  ; format: MP3 ; note: 5.8  ; id : 8
1 Titre: Travelin Man  Beautiful Loser ; Auteur: Bob Seger  ; format: AIF ; note: 1.3  ; id : 9
2 Titre: Smells Like Teen Spirit ; Auteur: Nirvana  ; format: WAV ; note: 8.6  ; id : 11
3 Titre: 96 Tears ; Auteur: Question Mark & The Mysterians  ; format: MP3 ; note: 0.3  ; id : 12
Fin Playlist  ---------------------
Numero de la playlist : 8
Chansons de la playlist :
0 Titre: White Rabbit ; Auteur: Jefferson Airplane  ; format: WAV ; note: 6.8  ; id : 100
1 Titre: Takin It To The Streets ; Auteur: Doobie Brothers  ; format: WAV ; note: 2.7  ; id : 1000
2 Titre: Interstellar ; Auteur: Frankie Rose  ; format: MP3 ; note: 9.1  ; id : 2000
3 Titre: Rainmaker ; Auteur: Traffic  ; format: MP3 ; note: 9.8  ; id : 3000
4 Titre: Rill Rill ; Auteur: Sleigh Bells  ; format: MP3 ; note: 4.4  ; id : 4000
Fin Playlist  ---------------------
```

FIGURE 7 – L'affichage du fichier précédent

5. lire toutes les playlistes qui suivent et les copier dans le fichier temporaire,
6. fermer le fichier de playlistes,
7. fermer le fichier temporaire, `.playlistes.tmp` ,
8. renommer le fichier temporaire avec le nom du fichier de playlistes (la fonction `rename` sera utile).

La fonction retournera 1 si la playliste existe, 0 sinon. Son prototype sera `int fdeletePlaylistNumber (char* fichier_playliste, int id) ;`

Remarque : avant d'appeler cette fonction, il faudra fermer le fichier ouvert, appeler la fonction puis ouvrir à nouveau le fichier de playlistes.

2.5 Fonctions utiles pour la réalisation des fonctions précédentes

Ecrire les fonctions suivantes :

— retourne la chanson d'identifiant `id` dans le fichier `fe`. Le prototype est `CHANSON rechercheCHANSON( FILE* fe, int id) ;`

— teste si la chanson dont l'identifiant est `id` existe et n'est pas vide dans le fichier `fe`. Une chanson vide ne contient que des zéros. La fonction `memcmp` sera utile à cet effet. Le prototype est `int existeCHANSON(FILE* fe, int id);`

— initialise la playliste `pl` en mettant tous les octets à 0. La fonction `memset` sera utile à cet effet. Le prototype est `void initPlaylist(PLAYLIST* pl) ;`

— affiche toutes les informations des chansons de la playliste `pl`. La playliste contient l'identifiant des chansons et il faut aller rechercher les informations sur les chansons dans le fichier `fichier_chanson`. Le prototype est `void printfPlaylist(PLAYLIST* pl,FILE* fichier_chanson) ;`

— lit au clavier toutes les informations de la playliste. La fonction stocke les informations dans la variable pointée par `pl`. Cette fonction vérifie que les numéros des chansons existent dans le fichier `fichier_chanson`. Le prototype est `void scanfPlaylist(PLAYLIST* pl, FILE* fichier_chanson) ;`

— compare 2 playlistes. La fonction retourne 0 si les playlistes sont égales, 1 sinon. Le prototype est `int cmpPlaylist(PLAYLIST* p1, PLAYLIST* p2);`

2.6 Les fonctions déjà écrites

Les fonctions suivantes pour la gestion d'un menu sont déjà disponibles sur le site des tp :

— dans les fichiers `interface.c` et `interface.h`
  — `int lit_choix_utilisateur( void )` : cette fonction affiche un menu avec 6 choix (Consultation, Modification, Destruction, Ajout, Afficher tout, Sortie) et retourne la valeur tapée au clavier qui doit être comprise dans les choix proposés.
  — `int lit_numero( void )` : cette fonction lit un numéro entier positif.
  — `void lit_chaine( char chaine[], int len )` : cette fonction lit une chaîne de `len` caractères au maximum au clavier, pouvant comporter des espaces. Le retour chariot est supprimé.
— dans le fichier `test_playliste.c`
  — le programme principal ci dessous que vous pouvez enrichir de la modification d'une playliste :

```
#include <stdlib.h>
#include <stdio.h>
#include "interface.h"
#include "music.h"
#include "playlist.h"

int main(int ac, char** ag) { int id; PLAYLIST p; FILE* fpl, *fch;
  // Usage de la commande : on lance avec les noms des fichiers de chansons et de playlistes
  if (ac!=3) {printf("Usage : %s chanson_binaire playlist_texte\n",ag[0]); exit(EXIT_FAILURE); }
    // Ouverture du fichier de chansons, en mode lecture binaire
  if ( (fch=fopen(ag[1],"rb"))==NULL) {
    fprintf( stderr, "Erreur d'ouverture du fichier %s\n", ag[1] );
    exit(EXIT_FAILURE);
  }
    // Ouverture du fichier texte de playlistes, en mode lecture/ecriture, position en fin de fichier
  if ( (fpl=fopen(ag[2],"a+"))==NULL) {
    fprintf( stderr, "Erreur d'ouverture du fichier %s\n", ag[2] );
    exit(EXIT_FAILURE);
  }
  while ( 1 ) {
      switch( lit_choix_utilisateur( ) ) {
        case CONSULTATION :
          if (fscanfPlaylistNumber(fpl,&p,id=lit_numero( ))) printfPlaylist(&p,fch);
          else printf("La playliste numero %d n'existe pas\n",id);
          break;
        case DESTRUCTION  : puts("Numero a detruire"); id=lit_numero( );
          fclose(fpl);
          fdeletePlaylist(ag[2],id);
          if ( (fpl=fopen(ag[2],"a+"))==NULL) {
            fprintf( stderr, "Erreur remplacement du fichier %s\n", ag[2] );
            exit(EXIT_FAILURE);
          }
          break;
        case AJOUT        :
          scanfPlaylist(&p,fch);
          fprintfPlaylist(fpl,&p);
          break;
        case TOUT         :
          fseek(fpl,0,SEEK_SET);
          while (!feof(fpl)) {
            fscanfPlaylist(fpl,&p);
            printfPlaylist(&p,fch);
          }
          break;
        case EXIT         :
          fclose( fpl );
          fclose( fch );
          return( EXIT_SUCCESS );
      }
    }
    return( EXIT_SUCCESS );
}
```