Dans cette série de TPs, nous allons programmer un utilitaire permettant d'effectuer un classement de fichier musicaux grâce aux informations contenus dans ces derniers. Nous étudierons le cas des fichier MPEG 1/2 Layer 3 (MP3) dont les informations sont stockées au format ID3v2.

Dans un premier temps, nous allons écrire le code permettant de lire ces informations. Vous trouverez sur moodle le fichier d'entête `tag.h` qui contient les définitions des structures que vous allez devoir utiliser ainsi que les prototypes des fonctions que vous devrez implémenter.

Des fichiers MP3 libres de droits et d'une musicalité sans faille sont disponibles sur `/home/infoens/hauspiem/public/asr4`. Vous pouvez les utiliser pour tester votre programme.

1 Description des tags au format ID3v2

Le format le plus répandu pour ajouter de l'information aux fichiers MP3 (comme l'interprète, le nom de l'album...) est ID3v2. Il consiste en une série d'octets placés en début de fichier.

1.1 Conventions d'écriture

Les conventions suivantes seront utilisées dans le TP :
- un nombre précédé du caractère `$` est un nombre hexadécimal dont l'octet de poids fort est indiqué en premier ;
- un nombre précédé du caractère `%` est un nombre binaire dont le bit de poids fort est indiqué en premier ;
- dans les tags ID3v2, tous les nombres sont stockés au format **BIG ENDIAN** ! Le microprocesseur de votre machine de TP est quant à lui **LITTLE ENDIAN**. Il faudra donc effectuer les conversions nécessaire.

1.2 Entête ID3v2

La première chose que l'on trouve dans le fichier est l'entête. Elle consiste en 10 octets répartis comme suit :

```
Identifiant ID3: "ID3"     -> 3 octets représentant la chaîne "ID3"
Version ID3v2:   $03 00    -> 2 octets représentant la version majeure (3) et mineure (0)
Flags ID3v2:     %abc00000 -> 1 octet dont les trois bits de poids fort représentent des options
Taille ID3:  4 * %0xxxxxxx -> 4 octets représentant la taille complète des tags.
                              Le bit de poids fort doit être ignoré.
                              Une taille de 257 sera représenté par les octets: $00 00 02 01
```

- Nous nous intéresserons uniquement à la version ID3v2.3.0. Les octets de version devront donc représenter une version inférieure ou égale à 3.0 ;
- Nous ne traiterons pas les flags pour des raisons de simplicité. Il faudra les lire dans le fichier, mais on ne s'intéressera pas à leur signification ;
- Le champ taille utilise un codage particulier. Quatre bits doivent être ignorés pour connaître la taille mais sont bien présents dans le fichier. Nous allons devoir effectuer des manipulations binaire (masques, décalage) pour reconstituer la bonne taille. La taille donnée correspond à la taille complète du tag ID3v2 **moins** la taille de l'entête.

1.3 Les *frames*

Chaque information est contenue dans une *frame*. Il y a donc plusieurs frames par fichier. Les frames sont disposées séquentiellement dans le fichier immédiatement après l'entête ID3v2. Une frame est constituée d'une entête de frame, suivie des données de la frame. Le format de l'entête de frame est le suivant :

```
Identifiant de Frame ID  $xx xx xx xx (quatre caractères)
Taille de frame          $xx xx xx xx (un entier sur quatre octets)
Flags                    $xx xx       (un entier sur deux octets)
```

L'identifiant de frame est une suite de quatre caractères normalisés. Par exemple, si la frame représente le nom de l'album, son identifiant sera `"TALB"`, si elle représente le numéro de la piste, son identifiant sera `"TRCK"`. Une frame n'est valide que si ces caractères sont des lettres majuscules (`'A'-'Z'`) ou des chiffres (`'0'-'9'`).

La plupart des frames possible sont de type *texte*, c'est à dire qu'elles représentent une chaîne de caractère et leur identifiant commence par la lettre `T`. Nous n'allons traiter que ce type de frame. À titre d'information, il peut également y avoir par exemple une frame dont l'identifiant est `"APIC"` qui contient une image correspondant à la piste musicale (en général, la couverture de l'album). Les données relatives à une frame commencent immédiatement à la fin de l'entête de frame (juste après le champ `flags`). Comme pour l'entête ID3v2, nous ignorerons également ces flags.

La taille d'une frame correspond au nombre d'octets **après** cette entête.

1.3.1 Frame de type texte

Les données d'une frame de type texte sont organisées de la façon suivante :

```
Format d'encodage    $xx
Valeur               <chaîne de caractère au format définit par l'encodage>
```

Le premier octet représente l'encodage utilisé pour la chaîne de caractère. Dans ID3v2.3.0, il en existe deux types.

- `$00` : la chaîne est encodée en `ISO8859-1`. C'est l'encodage basé sur la table ASCII qu'on utilise habituellement. Chaque caractère est codé sur un octet ;
- `$01` : la chaîne est encodée en Unicode 2.0. Le but de cet encodage est de coder les caractères dans des alphabets non latin. Chaque caractère est codé sur deux octets. L'ordre de ces octets ayant une importance, la chaîne commence par un marqueur (le BOM) qui indique l'ordre des octets. Ce marqueur sera `$FF FE` ou `$FE FF`. Traduire une chaîne Unicode pour l'afficher à l'aide d'un simple `printf` est compliqué. Nous effectuerons une conversion **simpliste** en utilisant le fait que les caractères latin sont simplement codés en utilisant leur code ASCII pour l'un des octets et zéro pour l'autre. L'ordre de ces deux octets est connu grâce au BOM. Si le premier octet du BOM est `$FF` alors le code ASCII du caractère latin sera le premier.

Immédiatement après l'octet d'encodage, on trouve les données de la chaîne.

Par exemple, si l'artiste (`TPE1`) d'une piste musicale est `"Muse"`, et que cette chaîne est encodée en `ISO8859-1` les octets représentant la frame seront :

```
$54 50 45 31   -> 'T'  'P' 'E' '1'
$00 00 00 05   -> Taille 5 octets
$00 00         -> Flags 0
$00            -> Encodage ISO
$4D 75 73 65   -> 'M' 'u' 's' 'e'
```

Si maintenant, on considère le même artiste, mais avec un encodage Unicode, la frame devient :

```
$54 50 45 31   -> 'T'  'P' 'E' '1'
$00 00 00 0b   -> Taille 11 octets
$00 00         -> Flags 0
$01            -> Encodage Unicode
$FF FE 4D 00 75 00 73 00 65 00   -> 'M' 'u' 's' 'e'
```

1.4 Octets de remplissage

Après les données des frames, le tag ID3v2 peut contenir un ensemble d'octets de valeur `$00` pour réserver de la place avant les données musicales en vue de la modification des tags. La taille donnée dans l'entête ID3v2 inclue ces octets.

1.5 Exemple complet

Voici le tag ID3v2 intégral d'une piste musicale dont les informations sont les suivantes :

- Album : "Origin Of Symmetry"
- Artiste : "Muse"
- Genre : "Rock"
- Titre : "New Born"
- Piste : "1"
- Année : "2001"

```
$49 44 33    -> la chaîne "ID3"
$03 00       -> Version majeure 3, mineure 0
$00          -> Flags à 0
$00 00 10 78 -> Taille: 120 + (16/2*256) = 2168 octets

$54 41 4c 42 -> Frame "TALB"
$00 00 00 13 -> Taille de frame 19 octets
$00 00       -> Flags de frame 0
$00          -> Encodage ISO
$4f 72 69 67 69 6e 20 4f 66 20 53 79 6d 6d 65 74 72 79 -> "Origin of Symmetry" (18 octets)

$54 50 45 31 -> Frame "TPE1"
$00 00 00 05 -> Taille de frame 5 octets
$00 00       -> Flags 0
$00          -> Encodage ISO
$4d 75 73 65 -> "Muse" (4 octets)

$54 43 4f 4e -> Frame "TCON"
$00 00 00 05 -> Taille de frame 5 octets
$00 00       -> Flags 0
$00          -> Encodage ISO
$52 6f 63 6b -> "Rock" (4 octets)

$54 49 54 32 -> Frame "TIT2"
$00 00 00 09 -> Taille de frame 9 octets
$00 00       -> Flags 0
$00          -> Encodage ISO
$4e 65 77 20 42 6f 72 6e -> "New Born" (8 octets)

$54 52 43 4b -> Frame "TRCK"
$00 00 00 02 -> Taille de frame 2 octets
$00 00       -> Flags 0
$00          -> Encodage ISO
$31          -> "1" (1 octet)

$54 59 45 52 -> Frame "TYER"
$00 00 00 05 -> Taille de frame 5 octets
$00 00       -> Flags 0
$00          -> Encodage ISO
$32 30 30 31 -> "2001" (4 octets)

$00 * 00     -> 2063 octets de valeur $00 (octets de remplissage)
```

---

2 Travail à effectuer

Le but principal de cette partie du TP est de vous apprendre à manipuler les fonctions d'entrées/sorties bas niveau. Vous allez donc avoir besoin d'utiliser les fonctions système `open`, `read`, `close` et `lseek` principalement.

Exercice 2.1 : Définition de types

Comme les données stockées dans un tag ID3v2 sont d'une taille précise, nous allons définir des types permettant de manipuler des entiers de taille connue.

Q 1. Définissez un type **u8** représentant un entier **non signé** de taille 1 octet.

Q 2. Définissez un type **u16** représentant un entier **non signé** de taille 2 octets.

Q 3. Définissez un type **u32** représentant un entier **non signé** de taille 4 octets.

Ces types doivent être définis dans le fichier `tag.h` qui vous est fourni. Il ne compilera pas sans eux, car ils sont utilisé dans les définitions des structures.

Exercice 2.2 : Fonctions de lecture de données

Le but de cet exercice est d'écrire les fonctions qui vont nous permettre de lire les informations situées dans le tag ID3v2. Nous allons avoir besoin de fonctions permettant :

- de lire des entiers de 1, 2 ou 4 octets depuis un fichier et de les convertir du format big endian au format little endian ;
- de convertir un entier de 4 octet en ignorant le bit 7 de chaque octet (pour la taille du tag ID3v2) ;
- de lire une chaîne de caractère d'une taille donnée depuis un fichier et de la convertir d'Unicode en ASCII si besoin.

Q 1. Implémentez la fonction :

```
int read_u8(int fd, u8 *val);
```

Cette fonction doit lire un entier stocké sur un octet depuis le fichier dont le descripteur est `fd` et affecter sa valeur à la variable pointée par `val`. La fonction doit retourner 0 en cas d'erreur et 1 en cas de succès.

Q 2. Implémentez la fonction :

```
int read_u16(int fd, u16 *val);
```

Cette fonction doit lire un entier stocké sur deux octets en format **big endian** depuis le fichier dont le descripteur est `fd` et affecter sa valeur à la variable pointée par `val` au format **little endian**.

Pour convertir un entier de deux octets du format big endian au format little endian, il suffit d'en inverser les octets (`$AABB` devient `$BBAA`).

La fonction doit retourner 0 en cas d'erreur et 1 en cas de succès.

Q 3. Implémentez la fonction :

```
int read_u32(int fd, u32 *val);
```

Cette fonction doit lire un entier stocké sur quatre octets en format **big endian** depuis le fichier dont le descripteur est `fd` et affecter sa valeur à la variable pointée par `val` au format **little endian**.

Pour convertir un entier de quatre octets du format big endian au format little endian, il suffit d'en inverser les octets (`$AABBCCDD` devient `$DDCCBBAA`).

La fonction doit retourner 0 en cas d'erreur et 1 en cas de succès.

Q 4. Implémentez la fonction :

```
u32 convert_size(u32 size);
```

Cette fonction convertit un entier de quatre octet (au format little endian) en un autre entier dont les bits 7 de chaque octet auront été ignorés. Par exemple :

```
%01101101 01110111 01101110 01101110
```

devient

```
%00001101 10111101 11110111 01101110
```

Cette conversion doit être effectuée à l'aide des opérateurs de décalage (`>>`), de **ET** bit à bit (`&`) et de **OU** bit à bit (`|`).

Q 5. Implémentez la fonction :

```
char *read_string(int fd, char *to, int size, int encoding);
```

Cette fonction permet de lire une chaîne de caractère de taille `size` octets depuis le fichier `fd` selon l'encodage `encoding` (0 pour ascii, 1 pour unicode). La fonction retourne alors une chaîne au format ascii, terminée par `'\0'`.

Si le paramètre `to` vaut `NULL`, la fonction alloue (à l'aide de `malloc`) l'espace mémoire nécessaire pour stocker la chaîne lue et retourne un pointeur vers cet espace.

Si le paramètre `to` ne vaut **pas** `NULL`, la fonction utilise l'espace mémoire pointé par `to` pour stocker la chaîne lue (elle suppose que cet espace est de taille suffisante). Elle retourne alors le pointeur `to`.

`NULL` doit être retourné en cas d'erreur.

Exercice 2.3 : Lecture des informations du tag

À l'aide des fonctions que nous avons implémentées, nous pouvons maintenant lire les tags depuis un fichier.

Q 1. Implémentez la fonction :

```
int tag_read_id3_header(int fd, id3v2_header_t *header);
```

Cette fonction prends en paramètre un descripteur de fichier ouvert en lecture `fd` depuis lequel lire l'entête ID3 et un pointeur vers une structure de type **id3v2_header_t** valide dans lequel la fonction doit stocker l'entête. Les champs seront lus **à l'aide des fonctions de l'exercice précédent**.

La fonction doit retourner -1 en cas d'erreur de lecture, si le fichier ne contient pas d'entête ID3v2 ou si elle est invalide ou dans une version supérieure à la version 2.3.0. Si tout s'est bien déroulé, elle retourne 0.

Q 2. Implémentez la fonction :

```
int tag_read_one_frame(int fd, id3v2_frame_t *frame);
```

Cette fonction lit une frame depuis le fichier `fd` et stocke les informations qu'elle contient dans la structure pointée par `frame`.

La fonction doit retourner -1 en cas d'erreur et 0 si tout s'est bien passé. En particulier, elle doit retourner -1 si elle rencontre une frame invalide [3].

La fonction ne doit gérer que les champs de type texte. Ces champs ont leur identifiant qui commence nécessairement par le caractère `'T'`. La seule exception est la frame `"TXXX"` dont vous devrez tout simplement ignorer la valeur de la même façon que les frames dont l'identifiant ne commence pas par `'T'`. La frame retournée dans ce cas aura son champ `text` positionné à la valeur `NULL`.

Après avoir lu la frame, la fonction doit positionner le fichier à la fin de celle-ci [4]

Exercice 2.4 : On colle les morceaux

Nous allons maintenant écrire une fonction qui ouvre un fichier musical, lit et vérifie l'entête ID3v2 et lit toutes les frames qu'il contient.

Q 1. Implémentez la fonction :

```
id3v2_frame_t *tag_get_frames(const char *file, int *frame_array_size);
```

Cette fonction doit ouvrir le fichier, lire l'entête et lire toutes les frames contenues dans le fichier. Afin de stocker les frames, la fonction devra allouer un tableau de frames. Comme on ne connaît pas à l'avance le nombre de frames, vous devrez d'abord allouer un petit tableau et augmenter sa taille au fur et à mesure de la lecture des frames [5].

La fonction s'arrête à la première frame invalide rencontrée. Si au moins une frame a été lue, la fonction retourne le tableau. Sinon, si aucune frame n'est présente ou si aucun tag ID3 n'est présent dans le fichier, la fonction doit retourner `NULL`

Q 2. Implémentez la fonction

```
void tag_free(id3v2_frame_t *frames, int frame_count);
```

qui libère la mémoire pointée par `frames` qui contient `frame_count` frames.

Q 3. Implémentez une fonction `main` qui prend en argument une liste de fichiers musicaux et qui affiche sur la sortie standard le nom de chaque fichier suivi des informations contenues dans le tag ID3v2 de ce fichier.

3. une frame invalide est telle que l'identifiant ne contient pas que des lettres majuscules ou des chiffres
4. encore à l'aide de la fonction `lseek`.
5. à l'aide de `malloc` et `realloc`