Maintenant que nous savons lire les informations depuis un fichier musical, nous allons pouvoir nous en servir pour organiser les fichiers selon un classement homogène.

1 Description du classement souhaité

Le résultat final du classement sera constitué de deux répertoires :
- `By Artist` : qui proposera un classement de morceaux par artiste/groupe ;
- `By Genre` : qui proposera un classement par genre.

Chaque fichier sera donc présent dans deux arborescences. Pour éviter de consommer deux fois la place nécessaire sur le disque, nous utiliserons des liens physiques. Le fichier aura donc deux noms, mais ces deux noms pointeront vers le même numéro d'i-nœud. Ainsi, les données du fichier ne seront présentes qu'une fois.

Pour le classement par artiste, l'arborescence devra refléter la suivante :

```
By Artist/
|--M
|  `-- Muse/
|      `-- Muse - Absolution/
|          |-- Muse - Absolution - 01.Intro to apocalypse please.mp3
|          |-- Muse - Absolution - 02.Apocalypse please.mp3
|          |-- ...
|          |-- Muse - Absolution - 13.Tsp.mp3
|          `-- Muse - Absolution - 14.Rule by secrecy.mp3
`--P
   `-- Pink floyd/
       `-- Pink floyd - Dark side of the moon/
           |-- Pink floyd - Dark side of the moon - 01.Speak to me.mp3
           |-- Pink floyd - Dark side of the moon - 02.On the run.mp3
           |-- ...
           |-- Pink floyd - Dark side of the moon - 08.Brain damage.mp3
           `-- Pink floyd - Dark side of the moon - 09.Eclipse.mp3
```

Pour le classement par genre, l'arborescence devra refléter la suivante :

```
By Genre/
`--Rock
   `-- Muse/
       |-- Muse - Absolution/
       |   |-- Muse - Absolution - 01.Intro to apocalypse please.mp3
       |   |-- Muse - Absolution - 02.Apocalypse please.mp3
       |   |-- ...
       |   |-- Muse - Absolution - 13.Tsp.mp3
       |   `-- Muse - Absolution - 14.Rule by secrecy.mp3
       `-- Pink floyd/
           `-- Pink floyd - Dark side of the moon/
               |-- Pink floyd - Dark side of the moon - 01.Speak to me.mp3
               |-- Pink floyd - Dark side of the moon - 02.On the run.mp3
               |-- ...
               |-- Pink floyd - Dark side of the moon - 08.Brain damage.mp3
               `-- Pink floyd - Dark side of the moon - 09.Eclipse.mp3
```

2 Travail à effectuer

Exercice 2.1 : Manipulation de chaînes

Nous allons écrire l'ensemble des fonctions permettant de générer les chaînes de caractères représentant les noms des fichiers et des répertoires.

Q 1. Implémentez la fonction :

```
const char *get_file_extension(const char *file);
```

Cette fonction prend en paramètre un nom de fichier et retourne son extension. Le pointeur retourné peut pointer vers le début de l'extension dans la chaîne passée en paramètre.

Q 2. Implémentez la fonction :

```
void clean_string(char *s);
```

Qui modifie la chaîne passée en paramètre de façon à *nettoyer* la chaîne. Le nettoyage consiste en :
- passer la première lettre en majuscule et les autres en minuscule (fonctions `toupper` et `tolower`) ;
- transformer le caractère `'/'` en caractère `'-'`. Ceci sera nécessaire pour ne pas générer de fichier contenant de `'/'` et donc ne pas poser de problème avec la création d'arborescence.

Exercice 2.2 : Récupération des informations

Pour manipuler facilement les informations des pistes musicales, nous allons utiliser une structure regroupant toutes les informations nécessaires :
- le nom de l'artiste ;
- le nom de l'album ;
- le genre ;
- le titre de la piste ;
- le numéro de la piste.

Q 1. Définissez une structure `sort_info_t` permettant de stocker les informations précédentes sous forme de chaînes de caractères.

Q 2. Implémentez la fonction :

```
int get_file_info(const char *source_file, sort_info_t *info);
```

Cette fonction prend en paramètre un nom de fichier et doit récupérer les informations ID3V2 qu'il contient et les stocker dans la structure pointées par le paramètre `info`.

Pour récupérer les tags ID3V2, vous utiliserez les fonctions du TP précédent. Les chaînes stockées dans la structure `info` devront être **nettoyées**.

Exercice 2.3 : Génération de l'arborescence

Nous allons maintenant nous intéresser à la création des répertoires.

Q 1. Implémentez les fonctions :

```
const char *get_artist_folder(char *buffer, int size, const char *root_folder, const sort_info_t *info);
```

et

```
const char *get_genre_folder(char *buffer, int size, const char *root_folder, const sort_info_t *info);
```

Ces fonctions doivent construire la chaîne de caractères représentant le répertoire dans lequel le fichier dont les infos sont données par `info` sera stocké. `root_folder` est le chemin vers le répertoire qui contiendra les répertoires `By Artist` et `By Genre`.

La chaîne construite devra être stockée dans `buffer` dont la taille maximale est `size` octets. Vous pouvez, par exemple, utiliser la fonction `snprintf`.

Les deux fonctions doivent retourner `buffer`.

Q 2. Implémentez la fonction :

```
int check_and_create_folder(const char *path);
```

qui créé un répertoire **s'il n'existe pas déjà** et vérifie que le processus possède bien les droits de lecture, écriture et exécution sur ce dernier. Si le répertoire existe déjà, seule la vérification de permission devra être faite. Vous devrez utiliser les fonctions `stat`, `access` et `mkdir`.

La fonction doit retourner 0 si le répertoire a été créé avec succès (ou s'il existe déjà) **ET** que les permissions sont bonnes. Dans tous les autres cas, la fonction devra retourner -1.

Q 3. Implémentez la fonction :

```
int create_tree(const char *fullpath);
```

qui s'assure que l'arborescence complète donnée par fullpath existe est que le dernier répertoire possède les droits en lecture, écriture et exécution. Elle réalise l'équivalent de la commande `mkdir -p`.

Cette fonction utilise bien évidemment la précédente. Elle doit retourner 0 si tout c'est bien passé et -1 si une erreur est survenue.

Exercice 2.4 : Tri final

Nous allons maintenant effectuer le tri d'un fichier.

Q 1. Implémentez la fonction :

```
const char *get_file_name(char *buffer, int size, const sort_info_t *info, const char *ext);
```

Cette fonction construit le nom du fichier qui sera généré en fonction des informations et de l'extension. La chaîne construite sera stockée dans `buffer`. Vous pouvez encore une fois utiliser la fonction `snprintf`. La fonction retourne `buffer`.

Q 2. Implémentez la fonction :

```
int sort_file(const char *root_folder, const char *source_file)
```

Cette fonction effectue le tri du fichier passé en paramètre. Les opérations qu'elle doit effectuer sont :

1. récupérer les informations ID3V2 contenues dans le fichier et vérifier que toutes les informations nécessaires pour le tri sont présentes (artiste, album, titre, numéro de piste et genre) ;
2. créer les répertoires de classement par artiste et par genre dans le répertoire `root_folder` ;
3. créer deux liens physiques (fonction `link`) du fichier source dans les deux répertoires précédemment créés. Le nom des liens physiques créés est obtenu à l'aide de la fonction précédente ;
4. si (**et seulement si**) la création des répertoires et des liens s'est déroulée correctement, supprimer le fichier d'origine (fonction `unlink`).

Exercice 2.5 : Programme principal

Vous allez maintenant écrire le main du programme. Les arguments de votre programme devront être les suivants :

```
$ musicsort repertoire_destination fichier1 [fichier2 ... fichierN]
```

Le premier paramètre est le répertoire dans lequel l'arboresence de tri va être créée. Les paramètres suivants sont des noms de fichiers et/ou de répertoires à trier. Si c'est un fichier régulier, on effectue le tri sur ce fichier. Si c'est un répertoire, celui-ci devra être parcouru **récursivement** et le tri appliqué à tous les fichiers réguliers qu'il contient.

Pour cela, vous allez devoir utiliser les fonctions `stat`, `opendir`, `readdir` et `closedir`. Si nécessaire vous pouvez écrire des fonctions supplémentaires pour vous aider à réaliser le `main`.