# I2010 : langage C (TP11)

Cet exercice est l'énoncé d'un ancien examen de 2019. Il avait une durée de 3h. La consigne principale était de compiler votre code avec les flags utilisés lors du cours :

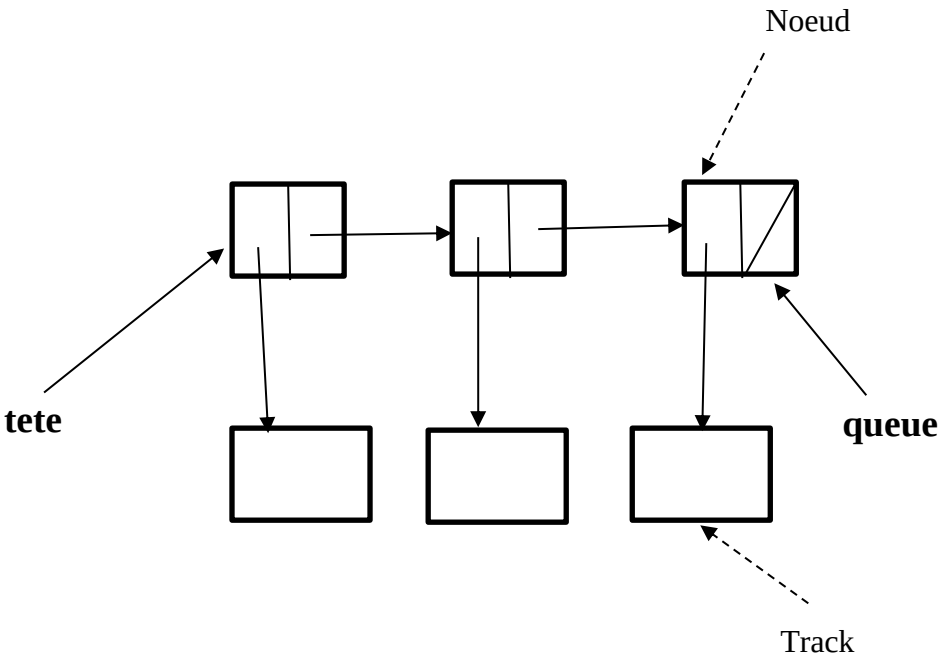
```
gcc -std=c11 -pedantic -Wall -Wvla -Werror
```

## Les structures récursives (suite et fin)

L'IPL voudrait implémenter son propre lecteur de fichiers audio. Pour ce faire, il vous est demandé d'implémenter certaines fonctionnalités permettant de gérer la liste de lecture.

## Hypothèses

- Les morceaux audio sont implémentés par un type *Track* contenant un nom d'artiste, un titre et une durée. Les champs titres et artistes sont de longueur limitée (voir *track.h*)
- Une liste de lecture correspond à un objet de type *Playlist* contenant deux pointeurs vers un *Noeud* : *tete* et *queue*. Dans une playlist vide, les deux pointeurs *tete* et *queue* sont NULL.
- Un *Noeud* contient un pointeur vers un *Track* appelé *track* et un pointeur suivant appelé *svt.*
- Le pointeur *svt* du dernier élément de la liste est NULL.



- Nous vous fournissons les fonctions *afficherTrack() et afficherPlaylist()*. Elles sont utilisées par le programme de test. Par conséquent ne les modifiez pas !

Voici un exemple d'affichage généré par la fonction *afficherPlaylist()* :

```
1 Happy Mondays - Loose Fit - 257
2 Panda Dub - Shankara - 403
3 Peter Tosh - Equal Rights - 360
4 Yves Simon - Amazoniaque - 260
```

## Fonctionnalités à implémenter

Nous vous fournissons les fichiers *track.h* et *track.c*, *playlist.h* et *playlist.c* ainsi que *testplaylist.c* à compléter.

### *Module Track: initTrack*

Cette méthode renvoie un pointeur vers une *Track* contenant la durée du morceau ainsi que la copie des chaines de caractères passées en paramètre ou NULL si la création n'a pas pu être faite. Si elles sont trop longues, ces chaînes seront tronquées à ARTISTE_LENGTH et TITLE_LENGTH.

### *Module Playlist : Définition de types*

Dans *playlist.h*, il vous est demandé d'écrire la définition des types *Nœud* et *Playlist*.

### *Module Playlist : initPlayList*

Crée une playlist avec les pointeurs *tete* et *queue* à *NULL*.

### *Module Playlist : ajouterTrack*

Ajoute un morceau en queue de liste.

### *Module Playlist : fusionDestructive*

La fonction *Playlist *fusionDestructive(Playlist *p1, Playlist *p2)* fusionne les playlists p1 et p2 en conservant l'ordre des tracks et en enlevant les doublons. De plus les listes p1 et p2 sont vidées. Elle renvoie donc une nouvelle playlist contenant les éléments des playlist p1 et p2. Après son exécution p1 et p2 sont vides (la tête et la queue sont nulls).

En cas de doublon, on libère la mémoire du *Nœud* qui n'est pas repris dans la liste, ainsi que celle du *Track* qu'il contient.

On suppose que chacune des deux listes fournies est triée, ne contient pas de doublon et que la liste fusionnée ne comporte pas non plus de doublon. Par conséquent, assurez-vous qu'un morceau apparaissant dans les deux listes n'apparaîtra qu'une seule fois dans la liste renvoyée.

Exemple :

```
Liste 1:
1 Happy Mondays - Loose Fit - 257
2 Panda Dub - Shankara - 403
3 Peter Tosh - Equal Rights - 360
4 Yves Simon - Amazoniaque - 260

Liste 2:
1 Eminem - Venom - 269
2 Karl Biscuit - La Morte - 236
3 Peter Tosh - Equal Rights - 360
Liste fusionnée:
1 Eminem - Venom - 269
2 Happy Mondays - Loose Fit - 257
3 Karl Biscuit - La Morte - 236
4 Panda Dub - Shankara - 403
5 Peter Tosh - Equal Rights - 360
6 Yves Simon - Amazoniaque - 260
```

### *Makefile*

Il vous est demandé d'écrire un *makefile* avec les différentes cibles pertinentes pour ce projet.

## Jeux de tests

Nous vous fournissons un programme *testplaylist.c* contenant quelques cas intéressants.

Notez que tous les cas limites de chaque fonction ne sont pas testés par cette classe. Libre à vous de modifier ce fichier pour ajouter des tests supplémentaires. Ce fichier ne sera pas corrigé.