INF1005C – PROGRAMMATION PROCÉDURALE
Travail dirigé no 5
Enregistrements et fichiers binaires

Objectif : Permettre à l'étudiant de manipuler des fichiers binaires et des enregistrements, ainsi que de maîtriser les concepts d'accès direct et séquentiel.

Durée : Deux séances de laboratoire.

Remise du travail : Avant 23h30 le 11 juin 2019.

Travail préparatoire : Lecture des exercices et de la documentation fournie et rédaction des algorithmes.

Directives : N'oubliez pas les entêtes de fichiers ni, aux endroits appropriés, les commentaires dans le code. Vous devez ajouter un en-tête pour chaque fonction. Dans l'écriture de l'entête d'une fonction, ne pas oublier de donner la description IN/OUT pour chacun des paramètres. Il est interdit d'utiliser les variables globales, sauf celles en lecture seule (constantes). Suivez en tout temps le guide de codage.

Documents à remettre : sur le site Moodle des travaux pratiques, vous remettrez l'ensemble des fichiers .cpp et .hpp compressés dans un fichier .zip en suivant la procédure de remise des TDs.

Mise en contexte

En audio numérique, la modulation d'impulsion codée (PCM en anglais) est une méthode utilisée pour représenter numériquement un signal analogique échantillonné. C'est ce qui est utilisé dans les ordinateurs, dans les CD audio et en téléphonie numérique. Dans un flux PCM, l'amplitude du signal analogique est enregistrée comme une valeur numérique à intervalles réguliers, c'est ce qu'on appelle l'échantillonnage. L'intervalle entre les échantillons est donné par la fréquence d'échantillonnage, qui est exprimée en Hz.

Un flux audio non compressé est donc composé d'une série d'échantillons de signal, et un fichier audio est typiquement composé d'un entête suivi d'un flux audio.

On vous demande de compléter un programme qui extrait les échantillons d'un fichier audio, d'appliquer certains effets de base et de réécrire les échantillons dans un fichier.

Dans le TD, on utilise des fichiers WAVE dont le format des échantillons est un LPCM stéréo sur 16 bits à une fréquence de 44.1 kHz. Cela signifie que chaque échantillon contient en fait deux échantillons, un pour la gauche et un pour la droite, chacun allant de

-32768 à 32767 (amplitude du signal) et qu'il y a 44 100 échantillons pour chaque seconde d'audio. C'est le format standard qui est utilisé dans les CD audio. Un fichier WAVE débute par un entête qui donne les caractéristiques du flux audio telles que la fréquence d'échantillonnage, le nombre de bit par échantillon, le nombre de canaux, etc.

Pour avoir une introduction plus détaillée sur l'échantillonnage et sur les fichiers WAVE, vous pouvez commencer par consulter Wikipédia :

fr.wikipedia.org/wiki/WAVEform_audio_format
fr.wikipedia.org/wiki/Modulation_d'impulsion_codée

Matériel fourni

Dans le fichier *CodeFourni.hpp* vous trouverez les déclarations des structures, constantes et variables qui vous sont fournies; leur implémentation se trouve dans *CodeFourni.cpp*. Vous devez écrire vos fonctions dans *CodeDemande.cpp* et mettre vos prototypes dans *CodeDemande.hpp*. Vous n'avez qu'à suivre les *TODO* dans chaque fonction. On vous donne aussi un fichier d'aide *Documentation.chm* qui contient toute la documentation du code fourni. Vous y retrouverez les descriptions détaillées des fonctions avec leur valeur de retour et paramètres ainsi que la description de ce que représentent les structures. Si jamais vous vous demandiez à quoi cela peut bien servir d'écrire des entêtes de fonctions, et bien en voici un très bon exemple!

On vous donne aussi cinq extraits de pièces musicale *schubert.wav*, *stravinsky.wav* et *varese.wav* pour effectuer vos tests. Ces fichiers sont encodés dans le format du TD.

Il y a plusieurs structures d'enregistrement avec lesquelles vous devez travailler. La structure `Echantillon` donne un échantillon stéréo (gauche et droite) en 16 bits. La structure `ListeEchantillons` est une séquence de taille dynamique de `Echantillon`, c'est-à-dire avec une taille, une capacité et un tableau d'éléments. Les structures `EnteteRiff` `EnteteFormat` et `EnteteDonnees`, regroupées dans `EntetesFichierWave` représente l'entête du fichier WAVE avec toutes les caractéristiques du flux audio. Consultez la documentation fournie pour plus de détails sur les structures, les constantes et les fonctions qui leur sont associées.

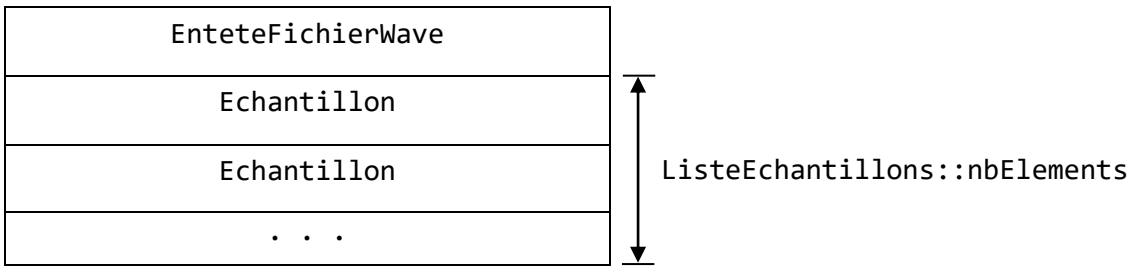
Présentation du travail à réaliser

Tout d'abord, lisez bien la documentation des fonctions fournies, car vous aurez à vous en servir. On vous fournit une fonction pour créer un `EnteteFichierWave` à partir d'une liste d'échantillons et des caractéristiques du format CD audio et une fonction pour calculer le nombre d'échantillon correspondant à un temps en secondes.

Vous avez onze fonctions à implémenter, certaines plus courtes que d'autres. On vous founit les squelettes dans *CodeDemande.cpp* et les prototypes dans *CodeDemande.hpp*. Pour chaque fonction, vous devez remplir les *TODO* qui vous sont donnés dans le code. Vous pouvez écrire plus de fonctions si vous le voulez, mais ce n'est pas nécessaire. Vous devez ensuite écrire le `main` de la même façon en suivant les *TODO*.

Il y a deux parties au TD. Les fonctions de la partie 1, c'est-à-dire de `ajouterElement()` à `ecrireFichierWave()` peuvent être faites durant la première séance de laboratoire. Les fonctions de la partie 2, c'est-à-dire `allouerListe()` et les autres après, demandent l'utilisation d'allocation dynamique qui est une notion vue entre la première et la deuxième séance. Vous devez ensuite vous servir de toutes vos fonctions dans le `main()` pour les tester.

Voici, en résumé, le format du fichier binaire :



N'oubliez pas d'écrire votre documentation!

Annexe 1 : Points du guide de codage à respecter

Les points du **guide de codage** à respecter **impérativement** pour ce TD sont les mêmes que pour le TD4 : (voir le guide de codage sur le site Moodle du cours pour la description détaillée de chacun de ces points)

Points du TD5 :

2 : noms des types en `UpperCamelCase`
3 : noms des variables en `lowerCamelCase`
5 : noms des fonctions en `lowerCamelCase`
21 : pluriel pour les tableaux (`int nombres[];`)
22 : préfixe *n* pour désigner un nombre d'objets (`int nElements;`)
24 : variables d'itération `i`, `j`, `k` mais jamais `l`
27 : éviter les abréviations (les acronymes communs doivent être gardés en acronymes)
29 : éviter la négation dans les noms
33 : entête de fichier
42 : `#include` au début
46 : initialiser à la déclaration
47 : pas plus d'une signification par variable
48 : aucune variable globale (les constantes globales sont tout à fait permises)
50 : mettre le & près du type
51 : test de 0 explicite (`if (nombre != 0)`)
52, 14 : variables vivantes le moins longtemps possible
53-54 : boucles `for` et `while`
58-61 : instructions conditionnelles
62 : pas de nombres magiques dans le code
67-78, 88 : indentation du code et commentaires
83-84 : aligner les variables lors des déclarations ainsi que les énoncés
85 : mieux écrire du code incompréhensible plutôt qu'y ajouter des commentaires
89 : entêtes de fonctions; y indiquer clairement les paramètres [out] et [in,out]