Travaux pratiques d'informatique musicale,

TP1 : Préparations aux TP

Licence 2, de physique et musicologie (version : 08/12/2024)

Frédéric Faure
Université Grenoble Alpes, France
frederic.faure@univ-grenoble-alpes.fr

Abstract

Dans ces **travaux pratiques** on utilise la librairie JUCE en C++ pour fabriquer des applications et plugins audio et midi.

Ce TP1 sert à préparer les TP suivants. A la fin de ce TP, vous aurez appris quelques notions de bases sur l'Audio et le MIDI, vous aurez préparé votre ordinateur pour les TP suivants, vous aurez créé des applications Audio et MIDI avec JUCE (une guitare numérique etc) et vous aurez appris les rudiments de la programmation en langage C++.

Dans la version électronique de ce document (pdf), vous accédez aux pages de cours et autres documents en **cliquant sur les liens de couleur qui entourent le texte**.

Le travail à faire en TP est indiqué sous la forme "Exercice (TP)". En préalable à la séance, on conseille de lire attentivement ce document, de faire les "Exercices (TD)". Ne pas hésiter à demander de l'aide technique à ChatGpt par exemple.

Contents

1 Les signaux Audio et MIDI

Dans cette première partie on résume des notion importantes à connaître pour la suite du TP.

1.1 Codage des nombres en base 2 (binaire) et base 16 (hexadécimal)

Video de cette section. Ce sont des notions simples mais importantes sur la représentation des nombres, utilisés en informatique en général et plus particulièrement utile en informatique musicale. Voir l'annexe B4 de ce **cours d'acoustique musicale**. Voir système binaire sur wikipedia.

1.1.1 La base 10 (décimale)

On rappelle qu'en base 10, qui est la base usuelle, on a 10 symboles disponibles: $0, 1, 2, \ldots, 9$ et que les nombres entiers sont écrits de la façon suivante:

$$0, 1, 2, \ldots 9, 10, 11, 12, \ldots, 19, 20, 21, \ldots$$

Definition 1.1. La règle est d'incrémenter la colonne de gauche lorsque l'on arrive au dernier symbole 9.

Exercise 1.2. En base 10, combien d'entiers différents peut-on écrire avec $B = 3$ cases $\square\square\square$ et un chiffre dans chaque case?

Solution 1.3. On peut écrire les entiers de $0 = 000$ à 999, ce qui fait $1000 = 10^3$ entiers différents. Plus généralement avec B cases, on peut écrire 10^B entiers différents.

1.1.2 La base 2 (binaire)

De la même façon, en base 2 on a 2 symboles disponibles, par exemple $0, 1$. Selon la même règle 1.1, les nombres entiers sont écrits de la façon suivante:

$$0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, \ldots$$

Exercise 1.4. En base 2, combien d'entiers différents peut-on écrire avec $B = 3$ cases $\square\square\square$ et un chiffre dans chaque case?

Solution 1.5. On peut écrire les entiers de $0 = 000$ à 111, ce qui fait $8 = 2^3$ entiers différents. Plus généralement avec B cases, appelé **nombre de bits**, on peut écrire 2^B entiers différents.

Remark 1.6. On rappelle qu'un **octet** correspond à 8 bits (i.e. 8 cases).

1.1.3 Le complément à 2

En base 2, avec un certain nombre de cases B fixé (nombre de bits), si on souhaite aussi **écrire des entiers négatifs**, on convient d'utiliser la première case pour le signe: 0 signifie positif et 1 signifie négatif. Par exemple avec $B = 3$ cases: on aura

entier	-4	-3	-2	-1	0	$+1$	$+2$	$+3$
code en base 2	100	101	110	111	000	001	010	011

- Remarquer que pour obtenir la deuxième ligne, on part de 0 et on remplit le tableau de façon croissante et circulaire.
- Plus généralement, avec B cases, on code les entiers de $-\frac{2^B}{2} = -2^{B-1}$ à $\frac{2^B}{2} - 1 = 2^{B-1} - 1$.
- L'avantage de cette convention est que l'opération d'addition est inchangée, par exemple pour calculer $+3 - 2$ on effectue en base 2: $011 + 110$ ce qui donne 1001, mais en fait 001 car on a que $B = 3$ cases. Le résultat est donc $+1$.

1.1.4 La base 16 (hexadécimale)

En base 16 on doit utiliser 16 symboles différents. La convention du système hexadécimal est d'utiliser dans l'ordre:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f$$

et on peut écrire les entiers en utilisant la règle 1.1. Comme $16 = 2^4$, il y a un avantage à utiliser la base 16 en informatique car ces 16 symboles sont codés avec 4 bits en base 2:

base 16	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
base 2	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
base 10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Ainsi un octet qui est 8 bits correspond à deux symboles en bases 16. On rappelle que deux symboles (x_1, x_2) en base 16 encodent le nombre $x_1 \times 16 + x_2$. Par exemple:

base 16	10	$a0$	ff
base 2	0001 0000	1010 0000	1111 1111
base 10	16	160	255

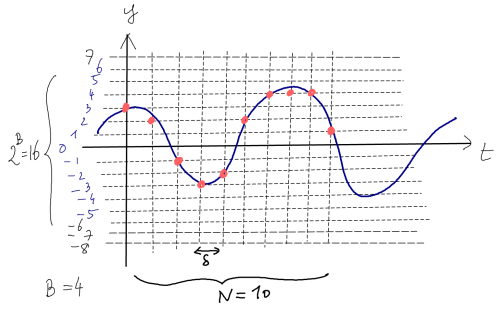
1.2 Signaux audio en informatique

Video des cette partie.

— Un **signal audio** est un signal sonore (une fonction $t \in \mathbb{R} \to s(t) \in \mathbb{R}$) qui a été transformée en une suite de nombre à l'aide d'une carte son (on parle de conversion analogique numérique).

— Pour cela on sélectionne des valeurs discrètes du temps t espacés par un petit intervalle $\delta = \frac{1}{N}$, avec par exemple $N = 44100$Hz intervalles par seconde, appelée fréquence d'échantillonage ou "rate frequency".

— A chacun de ces instants t, l'amplitude du signal $s(t)$ est approximée par un entier dans l'intervalle $(-2^{B-1}, 2^{B-1} - 1)$ où l'entier B est appelée "bit sampling", par exemple $B = 4$ sur la figure ci-dessous, mais habituellement on prend $B = 32$ pour avoir une bonne précision. Cette approximation est parfois appelée quantification du signal)



1.3 Signaux Midi en informatique

— Un **signal MIDI** est une suite de nombres qui représentent des instructions entre appareils musicaux (ex: claviers, surfaces de controle, expandeur, ordinateurs etc) en musique électronique.

Example 1.7.

— l'instruction de "jouer la note C de l'octave 5 avec un volume maximal" est représentée par la suite de nombres $\{144, 60, 127\}$ (car 144 signifie 'note on', 60 signifie 'C5' c'est à dire 'C' de l'octave 5, et 127 est le volume).

— l'instruction "éteindre la note C de l'octave 5" est représentée par la suite de nombres $\{128, 60, 127\}$ (car 128 signifie 'note off', 60 signifie 'C' de l'octave 5 et 127 signifie coupure nette du son).

— La liste des instructions MIDI est définie par la MIDI association et un résumé des messages importants que l'on étudiera plus tard.

Exercise 1.8. Ecrire les messages midi de l'exemple 1.7 en hexadécimal et en binaire.

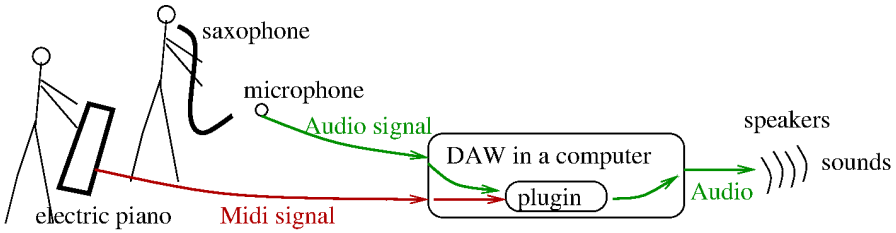
Solution 1.9.

Signification	Note on C5, vel=127	Note off C5, vel=127
décimal	$144, 60, 127$	$128, 60, 127$
hexadécimal	$90, 3c, 7f$	$80, 3c, 7f$
binaire	1001 0000 0011 1100 0111 1111	1000 0000 0011 1100 0111 1111

1.4 Logiciel DAW

— Un logiciel **DAW** (signifie Digital Audio Workstation) est un logiciel qui permet **d'enregistrer des signaux audio et midi sur des pistes** (tracks) et de **rejouer ces pistes**, puis envoyer le son sur des enceintes.

— Par exemple Ardour est un logiciel DAW open source et performant. Reaper: DAW simple et clair, il supporte tous les formats de plugins, mais n'est pas open-source. Dans le monde professionnel les ingénieurs du son utilisent fréquemment Protools sur windows PC ou Logic-Pro sur MacOs (mais ce sont des logiciels propriétaires).



1.5 Logiciel Plugin (ou Greffon ou Module)

— Un **audio plugin** est un logiciel qui peut être utilisé dans un DAW pour transformer des signaux audio et/ou MIDI.

— Voici des exemples de plugins où on précise la nature des signaux Entrée→Sortie:
  — Audio → Audio, sont des plugins appelés "AudioFX" (effets audio). Par exemple une "réverbération", un filtre fréquenciel.
  — Midi→Audio, sont des plugins appelés "instruments". Par exemple un synthétiseur.
  — Moins courants sont:
    — Midi→Midi, sont des plugins appelés "MidiFX" (Effet MIDI).
    — Audio,Midi→Audio, appelés "MusicEffect".
    — Audio→Midi, par exemple pour détecter des notes d'instruments acoustique et les envoyer vers un synthétiseur (voir par exemple Video on audio->midi).

1.5.1 Latence et buffer audio

La perception humaine de l'audio accepte des **retards, appelé latence, inférieur à** $L = 10\text{ms} = 0.01\text{s}$. **mais pas supérieurs**. Cela est du au temps de réponse des neurones dans notre cerveau.

Ainsi un plugin Audio comme ceux que nous réaliserons en TP nécessite de gérer des évènements MIDI et Audio et faire des calculs sur une durée assez courte, inférieure à cette latence de 10ms. On appelle cela du **'calcul en temps réel'.** Pour cela, le logiciel DAW envoit (et recoit en retour) au programme interne plugin les signaux MIDI et AUDIO de façon périodique par petits paquets appelés buffer.

Par exemple, nous avons vu en section 1.2 que la fréquence d'échantillonage est $N = 44100\text{Hz}$ échantillons par seconde. Donc dans la durée de latence acceptable $L = 10^{-2}s$. il y a

$$n = N \times L = 441 \text{ échantillons.}$$

Ainsi on demandera au DAW d'envoyer et recevoir des buffers audio de taille $n = 441$ à chaque période $L = 10^{-2}s$. ou quelque chose de comparable.

2 (TP) Installation préalable de logiciels

Video explicative.

2.1 C++

Dans les TP suivants nous utiliserons le langage de programmation C++. C'est un langage universel et très performant, bien adapté en particulier pour la conception de plugins audio, avec la librairie JUCE.

2.1.1 (Installation d'un compilateur pour C++

On aura besoin d'un environnement de programmation pour le langage C++. Selon votre ordinateur, suivre les instructions suivantes.

— Pour **Windows**: Installer **Visual-Studio** (option Community si il demande), avec le "Workloads Desktop development with C++", qui est un environnement de programmation pour coder en C++. Cela prend à peu près 10Go !

— Pour **MacOs**: Installer XCode.
— Pour **Linux**: Installer g++

2.1.2 Installation d'un éditeur pour le C++

On aura besoin d'un éditeur pour écrire le code C++. Installer Visual studio Code sous windows, linux et mac.

2.1.3 Installation de CMake

Installer CMake qui est un logiciel très utile pour compiler et développer des programmes sur plusieurs plateformes (pour ceux qui veulent approfondir, voici une présentation et voici un tutorial et references).

2.2 Pour construire des packages

2.2.1 Sous windows: Installation de NSIS

— Installer NSIS (Pour construire des packages)
— l'ajouter au PATH: Ouvre le Panneau de configuration : Va dans Système et sécurité > Système > Paramètres système avancés. Clique sur "Variables d'environnement". Dans la section "Variables système", trouve et sélectionne la variable Path, puis clique sur Modifier. Ajoute le chemin vers le dossier NSIS\makensis.exe (généralement C:\Program Files\NSIS\Bin
— Pour vérifier que NSIS est correctement installé et accessible : Ouvre une invite de commandes (cmd). Tape makensis et appuie sur Entrée. Si tout fonctionne, tu devrais voir l'aide pour NSIS.

2.3 Installation de JUCE (avec CMake)

Installer JUCE qui est un ensemble de bibliothèques C++ qui permettent de concevoir des plugins audio pour Linux, MacOs, IOS, Windows, Android, etc.
(Pour approfondir on peut suivre des tutorials de JUCE et le forum).

2.3.1 sous Windows

— Sur l'ordinateur, lancer Visual-Studio-Code, ouvrir un terminal avec Menu/View/Terminal et écrire (on peut faire copier/coller des lignes suivantes, les signes après # sont des commentaires qui n'ont pas d'effet):

```
cd $HOME; # se place dans le repertoire utilisateur
mkdir TP; # cree un repertoire
cd TP; # on se deplace dans le repertoire TP
```

— Sur le site de juce.com/download/, cliquer sur "Download from GitHub", puis bouton "Code" et "Download ZIP" et extraire le fichier téléchargé dans le répertoire TP. Cela crée le répertoire "JUCE-master". Reprendre le terminal et écrire les instructions suivantes (attention à bien respecter les espaces):

```
cd $HOME/TP;
cd JUCE-master;
mkdir build;
cd build;
cmake ../../JUCE-master -DCMAKE_INSTALL_PREFIX=JUCE
cmake --build .  --target install;
```

Cela installe et compile JUCE dans le répertoire $HOME/TP/`JUCE-master`/build/JUCE
— **Exemples Démos:** on compile (prépare) les démos en écrivant dans le terminal (attention à bien respecter les espaces):

```
cd $HOME/TP/JUCE-master
cmake  .   -B cmake-build -DJUCE_BUILD_EXAMPLES=ON -DJUCE_BUILD_EXTRAS=ON
cmake  --build cmake-build --target DemoRunner
```

On execute le programme de Démos en écrivant (ne pas oublier le point au début):

```
. $HOME/TP/JUCE-master/cmake-build/examples/DemoRunner/DemoRunner_artefacts/Debug/DemoRunner.exe
```

et essayer onglet: Browse Demos, et par exemple **Audio/PluckedStringsDemo** et cliquer sur les cordes de la harpe pour entendre le son. Prendre le temps d'essayer différents programmes de démonstration.

2.4 Donner des droits au répertoire VST3

— Clique droit sur le dossier C:\Program Files\Common Files\VST3.
— Sélectionne Propriétés, puis l'onglet Sécurité.
— Clique sur Modifier, choisis ton utilisateur, et coche la case "Contrôle total" pour lui donner les droits d'écriture.

2.5 Installation de DAWs

On peut installer un ou plusieurs DAW (Digital Audio Workstation). Voici quelques suggestions et remarques. Dans la suite on utilisera Ableton-Live comme exemple.

— Ableton Live. Disponible sur Windows et Mac. Seulement la version Demo est gratuite. Très utilisé par les musiciens.
— **Cakewalk by BandLab**. Disponible sur Windows. Gratuit.
— Ardour. Disponible sur Windows et Mac et Linux. Gratuit (ou 1 euro ou plus).
— **GarageBand**. Disponible sur Mac. Gratuit. Options limitées.
— **Audacity.** Disponible sous Windows et Mac et Linux. Gratuit

2.6 Sous Windows choix d'un pilote audio sans latence (optionnel)

Fais un clic droit sur le bouton Démarrer (ou appuie sur Windows + X), puis sélectionne Gestionnaire de périphériques dans le menu. Contrôleurs audio, Identifier le pilote audio : Fais un clic droit sur le périphérique audio et sélectionne Propriétés pour **lire le nom du pilote audio**. Par exemple Realtek Audio. Si il y a de la latence, installer **ASIO4ALL**, qui est une solution gratuite pour **améliorer la latence audio sur Windows**. Demander de l'aide à ChatGpt.

3 (TP) Initiation au langage C++

Video de cette partie.

Le C++ est un langage informatique qui permet de donner des instructions, des taches précises à un ordinateur. La conception de plugins audio/midi nécessitent des concepts assez évolués d'informatique comme la **"programmation temps réel"** ce qui signifie que l'on doit contrôler la durée d'une tache, par exemple inférieure à 1ms, **"multi thread"** ce qui signifie que l'on doit avoir différents programmes qui tournent en parallèles et communiquent entre eux, **"multi plateformes"** ce qui signifie que le programme doit pouvoir produire des plugins audio pour plateformes Mac-Os, windows, linux et eventuellement Ios, Android, etc et **"programmation objet en C++"**, ce qui signifie que ce langage utilise des concepts de "classes d'objets".

Cependant le but de ces TP est avant tout de réaliser un projet intéressant d'un point de vue physique, mathématique et musical en s'initiant à la programmation. Pour cette raison, **on ne demande pas de maîtriser à priori les concepts d'informatiques, on présentera leur logique au fur et à mesure.** Il faut savoir que ces concepts ne sont pas compliqués d'un point de vue logique mais sont souvent compliqués en apparence car leur syntaxe est riche et suit des conventions très précises (l'informatique actuelle ne supporte pas la moindre erreur de syntaxe sinon le programme bug). Cependant les IA récentes comme ChatGpt sont d'une grande aide pour s'initier et pratiquer la programmation.

On propose ici quelques exercices d'initiation à la programmation en C++ qui n'aura aucun rapport avec JUCE et les plugins qui sont les sujets dans la suite des TP.

— Pour une initiation à C++ on propose le tutorial ou celui ci: tutorial C++ orienté physique numérique

Un programme en C++ est une suite d'instructions à destination de l'ordinateur. Elles sont dans une langue informatique appelée C++ que l'humain comprends et que l'ordinateur devra traduire dans son propre langage ("langage machine) dans une deuxième étape appelée "compilation". A la fin on aura un fichier appelé "executable".

Exercise 3.1. (TP) **"Ecrire, compiler et executer un premier programme C++ très simple"**

1. Pour **écrire un programme C++**, ouvrir "Visual studio code" (VSC). Faire File/New_File et appeler ce nouveau fichier "`salut.cc`" (le suffixe .cc ou .cpp signifie C++) que l'on place dans le répertoire `$HOME/TP/TP1/salut.cc` . Dans ce fichier, écrire le code suivant, où le texte après // sont des commentaires:

```
   #include <iostream> // Inclusion de la bibliothèque pour les entrées/sorties
vers l'écran
   using namespace std; // Evite d'ecrire std::cout ci-dessous

   int main() // Ligne de départ du programme
   {
   cout << "Salut!" << endl; // Affiche "Salut!" suivi d'un retour à la ligne
   return 0; // Retourne 0 pour indiquer que le programme s'est terminé correctement
   }
```

2. Pour **compiler le programme précédent avec cmake**, toujours avec VSC écrire le code suivant dans le fichier `$HOME/TP/TP1/CMakeLists.txt`

```
   #- fichier d'instructions pour cmake pour le script de compilation suivant
   cmake_minimum_required(VERSION 3.10) # version minimale de CMake
   project(TP1_salut) # Nom du projet
   add_executable(salut salut.cc) # pour créer executable salut à partir du
fichier salut.cc
```

   Ecrire le code suivant dans le fichier `$HOME/TP/TP1/compile.ps1`

```
   #- script pour compiler
   cd $HOME/TP/TP1/ #changement de répertoire
   mkdir build # crée répertoire
   cd build # changement de répertoire
   cmake ..  # génère les fichiers
   cmake --build .  --config Debug # compile en mode Debug
```

   Pour compiler, executer le script précédent: dans VSC, ouvrir un Terminal avec "Menu/View/Terminal" et écrire (ne pas oublier le point au début):

```
   .  $HOME/TP/TP1/compile
```

   Résultat: l'ordinateur a fabriqué un fichier executable: `salut.exe`

3. **Executer le programme:** dans VSC, ouvrir un Terminal avec "Menu/View/Terminal" et écrire (ne pas oublier le point au début):

```
   .  $HOME/TP/TP1/build/Debug/salut.exe
```

   Résultat: le programme affiche dans le terminal:

```
   Salut!
```

On a appris comment mettre en oeuvre l'écriture, la compilation et l'execution d'un programme en C++. On va maintenant apprendre quelques bases du langage C++ afin de comprendre la logique de la programmation. On continue à utiliser VSC.

Exercise 3.2. (TP) **"Quelques opérations de base en C++"**

1. Ecrire ce fichier C++ `$HOME/TP/TP1/calculs.cc`

```
   #include <iostream> // Inclusion de la bibliothèque pour les entrées/sorties
vers l'écran
   using namespace std; // Evite d'ecrire std::cout ci-dessous

   int main() // Ligne de départ du programme
```

```
{
int a,b; // déclaration des objets a et b de la classe int
a=1; // affectation:  a prend la valeur 1
b=a+1; // affectation b prend la valeur 2
int c; // déclaration de l objet c de classe int
c=a+b; // affectation:  c prend la valeur a+b c'est à dire 3
// affichage à l'écran:
cout << "la somme de " << a << " et " << b << " vaut " << c <<endl;
}
```

Dans le fichier `CMakeLists.txt` ajouter la ligne:

```
add_executable(calculs calculs.cc)
```

Executer le nouveau programme `calculs.exe` qui affichera:

```
la somme de 1 et 2 vaut 3
```

2. La classe `double` permet de stocker des **nombres à virgules** (appelés "avec double précision"). La ligne `#include <math.h>` permettra d'utiliser des fonctions de mathématiques. Ajouter quelques lignes au programme `calculs.cc` afin de déclarer l'objet x=3.14 et calculer y=sin(x) puis afficher y.
3. Dans ce cours de C++, lire la section 3.2.3 sur la classe **vector** qui permet de stocker des listes, copier les exemples de programmes et les executer.
4. Dans ce cours de C++, lire la section 4.1 sur la **"boucle for"** et faire l'exercice 4.1.2
5. Dans ce cours de C++, lire la section 5.1 sur la notion de **"fonction".**
6. Optionnellement, on suggère d'étudier les premiers chapitres de ce cours de C++,