LAPINS NOIRS

- La Caverne Aux Lapins Noirs -

- La Caverne aux Lapins Noirs -

Lors de cette ruée, vous allez programmer votre premier logiciel audio.
Un séquenceur de son est une machine servant à réaliser une unique mélodie configurable

Ce document est strictement personnel et ne doit en aucun cas être diffusé.

Auteur : Jason Brillante

INDEX

01 – Avant-propos

Votre travail doit être rendu via les dossiers **~/rue/sequencer/** dans votre espace personnel.

Si vous faites erreur et que le dossier que vous utilisez pour votre rendu est différent, vous ne serez pas évalué faute d'avoir pu trouver votre travail.

Ce travail est a effectuer en binôme. Vous pouvez bien sur échanger avec vos camarades, néanmoins vous devez être l'auteur de votre travail. Utiliser le code d'un autre, c'est **tricher**. Et tricher annule **toutes** les médailles que vous avez reçu sur l'activité. La vérification de la triche est réalisée de la même manière que la correction : de manière **automatique**. Prenez garde si vous pensez pouvoir passer au travers.

Votre rendu doit respecter **strictement** l'ensemble des règles suivantes :

- Il ne doit contenir **aucun** fichier objet. (*.o)
- Il ne doit contenir **aucun** fichier tampon. (*~, #*#)
- Il ne doit pas contenir votre production finale (programme ou bibliothèque)

La présence d'un fichier interdit mettra immédiatement fin à votre évaluation.

Votre programme doit respecter les Tables de la Norme dans leur intégralité. Vous êtes invité à les observer depuis **l'Infosphère**. Elles sont disponibles comme ressource de cette activité.

02 – Fonctions autorisées

La bibliothèque logicielle venant avec le C est vaste et disponible. La LibLapin, que vous utilisez dans vos projets multimédia, est également vaste... Cependant nous avons fait le choix de vous interdire leurs utilisation intégrales, afin de vous amener progressivement à reprogrammer vous même ses fonctionnalités les plus utiles.

L'utilisation d'une fonction interdite est assimilée à de la triche. La triche provoque l'arrêt de l'évaluation et la perte des médailles.

Vous n'avez le droit d'utiliser aucune fonction issue de la LibC ou de la LibLapin à l'exception de celles que nous vous autoriserons explicitement.

Pouvoir utiliser une fonction ne signifie pas nécessairement que celle-ci soit utile à votre cas.

Pour cette activité, issu de la LibC, vous n'avez le droit qu'à la liste suivante :

- open
- close
- read
- srand
- cos
- atan2
- write
- alloca
- atexit
- rand
- sin
- sqrt

Remarquez bien l'absence de **malloc** et de **free**. En effet ils sont **interdits** ! Issu de la LibLapin, vous avez le droit aux fonctions suivantes :

- bunny_start
- bunny_stop
- **bunny_malloc**
- **bunny_free**
- bunny_new_pixelarray
- bunny_delete_clipable
- bunny_blit
- bunny_display
- bunny_set_*_function
- bunny_set_*_response
- bunny_loop
- bunny_create_effect
- bunny_compute_effect
- bunny_sound_play
- bunny_sound_stop
- bunny_delete_sound
- bunny_open_configuration
- bunny_delete_configuration
- bunny_configuration_getf
- bunny_configuration_setf
- bunny_configuration_*
- bunny_set_memory_check
- bunny_release
- bunny_usleep

La suite sur la page d'après.

Pour utiliser **bunny_malloc**, vous pouvez soit programme directement avec, soit en utilisant le modèle de projet qui vous a été transmis, mettre **1** dans la variable de Makefile **BMALLOC**, qui transformera **malloc** en **bunny_malloc**.

Vous appellerez **bunny_set_memory_check** au début de votre fonction **main** de sorte à provoquer une vérification de vos allocations à la fermeture du programme.

```
bunny_malloc, par défaut, limitera votre consommation de RAM
                        à 20Mo.

                    Pour information :
            Une image en 1920*1080 fait environ 8Mo.

  Une musique en 44kHz de 1 minute en stéréo fait environ
                        10Mo.

  Vous devrez donc disposer d'une discipline de fer avec vos
    allocations... et probablement trouver des compromis.
```

L'utilisation de **bunny_malloc** parfois **cachera** des erreurs dans votre programme, et parfois en **révélera** : son principe d'allocation était différent de **malloc**, il sera parfois plus « fort » ou plus « faible ». Ne vous mentez pas à vous en disant « l'utilisation de **bunny_malloc** fait planter mon programme », ce n'est pas **bunny_malloc**, c'est *vous*.

N'hésitez pas à l'activer, à le désactiver (Et lorsqu'il est désactivé, à utiliser valgrind). Et n'oubliez pas que désormais, votre demande de RAM a de véritables chances d'échouer. Car exploiter aussi peu de RAM, cela va très vite…

Dans votre rendu, il ne devra pas y avoir la moindre trace de malloc.

06 – Créer un son et le jouer

La fonction **bunny_new_effect** de la LibLapin permet de créer à partir d'une durée en seconde un effet sonore. Cet effet sonore contient un certain nombre d'échantillon dont la quantité est accessible via l'attribut **sample_per_second** présent dans **t_bunny_effect**.

Le champ **sample** du **t_bunny_effect** est un tableau d'entier sur 16 bits. Chaque valeur indique un niveau dans l'onde sonore.

Vous pouvez écrire les valeurs que vous souhaitez dans **sample[]**. Pour demander à votre système de transmettre l'onde sonore que vous avez écrit à la carte son, vous devrez appeler **bunny_compute_effect**. Pour jouer votre son, la fonction **bunny_sound_play** vous permettra de la jouer.

Pour générer des ondes simples, vous pouvez utiliser les fonctions **sin**, **cos**, **tan** et **rand**. Bien sur, vous pouvez aussi inventer des formes d'ondes plus complexes…

Pour l'instant, essayer simplement de faire un programme qui génère un son continu, n'importe lequel. Ensuite, vous essayerez de créer un programme faisant un LA international.

Qu'est ce qu'un LA international ? Rendez vous sur **Wikipedia**. Documentez-vous.

Une fois que vous aurez trouvé comment générer cette fameuse note, pourquoi ne pas écrire la fonction suivante ?

```
void std_sing(t_bunny_effect        *sound,
              size_t                start,
              size_t                end,
              size_t                frequency);
```

Start indique le début de la note à jouer, **end** la fin. **frequency** la note elle-même. Utilisez **sin** pour générer votre onde. **L'onde doit s'ajouter au sons existants.**

07 – Le séquenceur

Un séquenceur est un instrument de musique électronique dont l'utilité est de jouer en boucle de courtes mélodie configurable en hauteur et en tempo.

Voici un exemple de séquenceur en fonctionnement, jouant « Der mussolini » de Deutsch-Amerikanische Freundschaft, plus connu (mais tout de même modestement) sous le nom de D.A.F.

https://www.youtube.com/watch?v=BVHJWTX_gIo

Votre programme fonctionnera de la manière suivante, au départ du moins :

```
$> ./sequencer tempo frequencies+
```

Il jouera en boucle la mélodie décrite dans les paramètres. Ces paramètres sont le tempo (c'est à dire le note de temps par minute, c'est à dire le nombre de note dans notre cas) ainsi que les notes elle-même, présente ici sous la forme de fréquences.

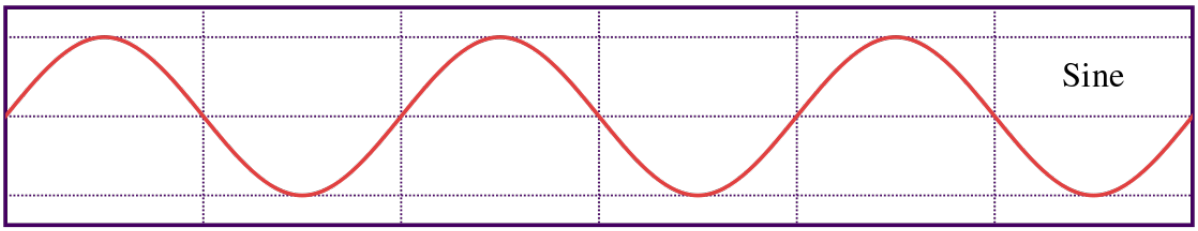
Que mettre comme valeur ? Comment faire un LA ? Rendez vous sur Wikipedia.

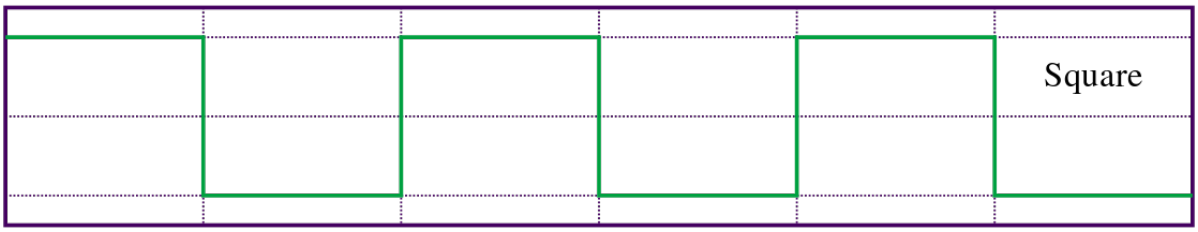
A ce stade de la ruée, la forme d'onde attendue est la sinusoide.

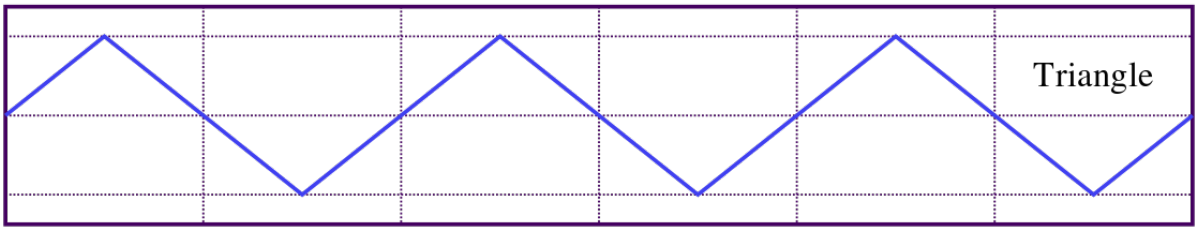
08 – Le synthétiseur

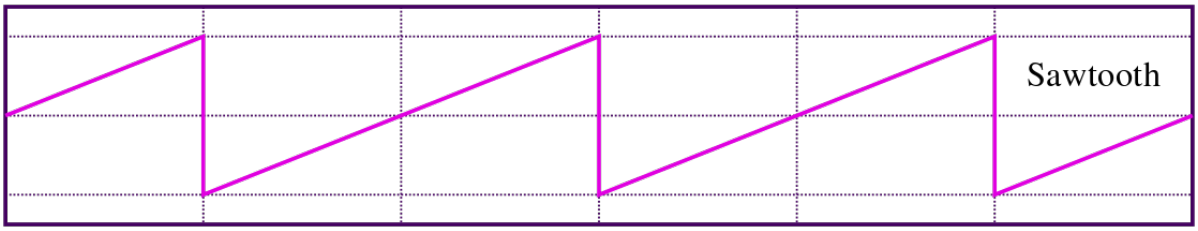
Le nouveau premier paramètre du programme est maintenant la forme de l'onde sonore que vous générez.

Voici les ondes que vous devriez générer : **sin**, **square**, **triangle**, **saw**. Vous êtes libres d'en créer d'autres. **tan** est intéressante aussi, mais il y a bien sur des combinaisons à faire…



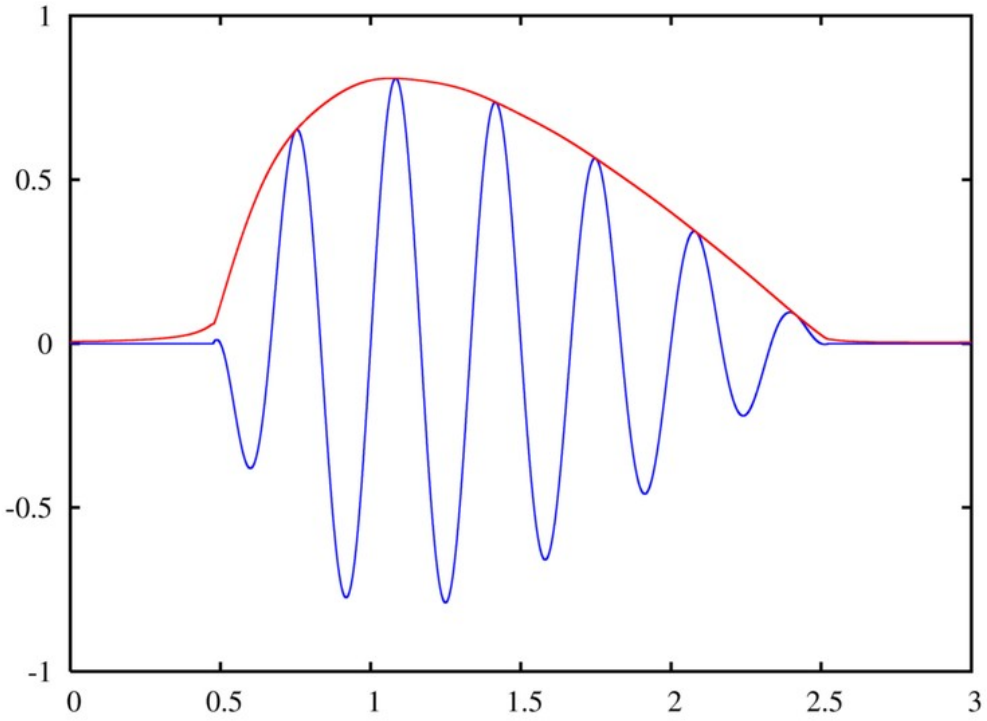






Ce paramètre peut-être double : l'ajout d'un ' :' à la suite de la forme de l'onde permet de spécifier une enveloppe.

Une enveloppe permet de modifier le volume du son lors de sa durée de vie.



Voici un exemple syntaxique, un son en dent de scie dans une enveloppe triangulaire : « saw:triangle ». Si aucune enveloppe n'est spécifiée, alors il n'y en a pas, (ou elle est assimilée à une enveloppe d'onde carrée…)

Pourquoi ne pas rajouter un multiplicateur ensuite ?

Par exemple si l'on fait « saw:trianglex4 », alors l'enveloppe triangle fera 4 cycles au lieu d'un seul. A de faibles multiplications, cela donnera l'impression d'une augmentation de tempo, avec de haute multiplication, cela changera beaucoup la texture du son.

09 – Slide

Le slide est un effet de glissement allant d'une note à l'autre : au lieu de passer d'une note à l'autre directement, l'onde va s'accélérer ou ralentir progressivement de manière à ce que toutes les notes intermédiaires entre deux fréquences soient jouées.

Pour signifier qu'un slide doit avoir lieu, à la fin d'une fréquence, le symbole **':'** peut-être ajouté. Dans ce cas, cela signifie que la note est joué et qu'ensuite on glisse directement à la note d'après.

Le « glide » est un effet consistant à effectuer cet effet entre toutes les notes joués.

Il est fréquent sur les puces synthétiseurs primitives de disposer en plus de la possibilité de glisser vers la prochaine note, de la possibilité de glisser vers la même note à l'octave du dessus. Ajoutez la possibilité d'utiliser le symbole '~' à l'arrière d'une fréquence, pour provoquer cet effet.

De la même manière qu'avec le multiplicateur d'enveloppe, les symboles ':' et '~' peuvent être suivis d'un multiplicateur permettant de répéter l'opération de slide plusieurs fois au lieu d'une unique fois, cela sur un même temps.

10 – L'instrument

Un instrument de musique peut-être entièrement automatique, mais n'est il pas intéressant de pouvoir aussi jouer dessus ?

Ajouter une interface graphique à votre séquenceur. L'objectif de celle-ci sera de remplacer les paramètres. Vous pouvez tout de même en conserver certains. Vous êtes libre de réaliser l'interface de votre choix.

Comment la rendre jolie sans avoir trop à programmer ?
Dessinez une image et afficher là.
Ne programmez que le graphisme des éléments mobiles.

Utilisez de préférence la souris.