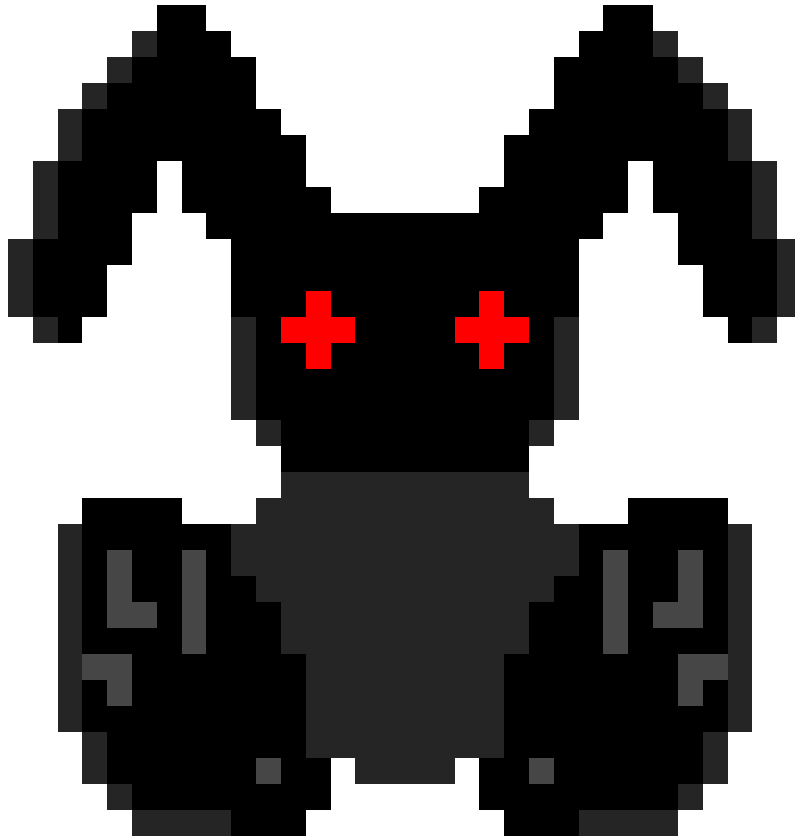




Programmation audio



# LAPINS NOIRS

ONDE SONORE

Soyez prêt à baisser le volume

- La Caverne aux lapins Noirs -

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



# INDEX

- 01 – Avant-propos
- 02 – Fonctions autorisées
- 03 – Méthode de construction
  
- 04 – Accéder à la mémoire audio
- 05 – Générer une onde simple
- 06 – Des ondes diverses



## 01 – Avant-propos

Votre travail doit être rendu via le dossier `~/tp/wave/` 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 à effectuer seul. Vous pouvez bien sûr 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  | - write  |
| - close | - alloca |
| - read  | - atexit |
| - srand | - rand   |
| - cos   | - sin    |
| - atan2 | - 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_set_*_function | - bunny_open_configuration   |
| - bunny_stop            | - bunny_set_*_response | - bunny_delete_configuration |
| - <b>bunny_malloc</b>   | - bunny_loop           | - bunny_configuration_getf   |
| - <b>bunny_free</b>     | - bunny_create_effect  | - bunny_configuration_setf   |
| - bunny_new_pixelarray  | - bunny_compute_effect | - bunny_configuration_*      |
| - bunny_delete_clipable | - bunny_sound_play     | - bunny_set_memory_check     |
| - bunny_blit            | - bunny_sound_stop     | - bunny_release              |
| - bunny_display         | - bunny_delete_sound   | - bunny_usleep               |

La suite sur la page d'après.



Pour utiliser **bunny\_malloc**, vous pouvez soit programmer 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**.



## 03 – Méthode de construction

Il peut vous être demandé d'écrire des programmes ou des fonctions.

Dans le cas des programmes, il vous sera toujours demandé de fournir un **dossier** pour l'exercice le requérant. Un **Makefile** vous sera également demandé. Le **nom du programme** de sortie vous sera précisé à chaque fois. Un Makefile incorrect, un mauvais nom de programme, et votre correction n'aura pas lieu...

Dans le cadre des fonctions, il vous ai demandé de fournir le fichier dans votre **dossier de bibliothèque personnelle**, de sorte à ce que vous puissiez utiliser toutes les fonctions que vous avez déjà réalisé jusqu'ici. Pour rappel, le dossier de votre bibliothèque doit être placé à la racine de votre espace personnel et s'appeler **libstd/**.

N'oubliez pas d'entretenir avec soin votre dossier **libstd/** de sorte à ce qu'il soit toujours propre, respecte la norme et soit en état de compiler... sans quoi elle fera obstacle à la correction.

Votre compilation devra toujours comporter les options **-W**, **-Wall** et **-Werror**.

Dans le cadre de la programmation multimédia, le système de correction établira toujours la variable d'environnement **BMALLOC** à 1. Si vous utilisez le modèle de projet, cela provoquera l'utilisation de **bunny\_malloc** dans votre bibliothèque personnelle comme dans votre projet rendu.



## 04 – Accéder à la mémoire audio

Pour créer un espace dans la carte son, utilisez la fonction `bunny_new_effect` qui prend en paramètre une durée du son en seconde. Cette durée est un flottant, cela signifie qu'il est possible demander un nombre de seconde non rond.

La mémoire renvoyée, de type `t_bunny_effect` contient de nombreux champs dont le plus intéressant est `sample`. Ce champ est similaire au champ `pixels` du `t_bunny_pixelarray` dans le sens où c'est **la donnée qui sera exploitée**. La longueur de l'espace mémoire pointée par `sample` est `duration` multiplié par `sample_per_second`.

Si `duration` n'a certainement aucun secret pour vous : c'est le paramètre passé à `bunny_new_effect`. Le champ `sample_per_second` est par contre probablement inconnu : il s'agit de la fréquence d'échantillonnage, c'est à dire le nombre de niveau que l'onde sonore peut prendre en une seconde. Plus cette valeur est élevée, plus le son est de bonne qualité et des nuances peuvent être entendues.

Par défaut, la fréquence d'échantillonnage est 44100Hz, c'est à dire qu'il faut écrire dans 44100 cases de `sample` pour faire un son d'une seule seconde... Chaque niveau enregistrée l'est dans un entier de 16 bits, c'est à dire un entier dont les valeurs possibles sont comprises entre -32768 et 32767. Plus l'amplitude – la valeur absolue des valeurs formant l'onde sonore – est grande, plus le volume est important.

Vous allez commencer simplement en réalisant une **friture**, de la même manière que celle que vous deviez réaliser **graphiquement** : remplissez `sample` de valeurs aléatoires à l'aide de la fonction `rand`. N'hésitez pas à tenter d'exploiter toute l'amplitude disponible, entre -32768 et 32767, mais *faites attention à mettre le volume à un niveau raisonnable avant de tester*.

```
void      std_set_noise(t_bunny_effect *fx) ;
```

Pour jouer le son, une fois que vous l'avez écrit, vous devrez d'abord appeler `bunny_compute_effect` qui permet de faire passer le son que vous avez écrit à la carte son. Ensuite, vous pourrez appeler `bunny_sound_play`. Vous remarquerez que `bunny_sound_play` prend un pointeur sur `t_bunny_sound` et non un pointeur sur `t_bunny_effect`... mais ce n'est pas grave, car il y a un `t_bunny_sound` dans `t_bunny_effect` ! Envoyez simplement son adresse et préparez vous à danser avec modération.



## 05 – Générer une onde simple

Programmez la fonction suivante :

```
void std_set_wave(t_bunny_effect *fx,
                 int start,
                 int stop,
                 int frequency);
```

Cette fonction remplit depuis la case **start** jusqu'à la case **stop non compris** la mémoire son de **fx** d'une onde sonore allant à la fréquence **frequency**.

Qu'est ce que cette fréquence signifie ? La fréquence d'un son détermine sa note : plus la fréquence est élevée, plus la note sera aiguë. Une fréquence de 100 indique que en **une seconde, il y a 100 ondulations**. L'acuité humaine est généralement comprise entre 20Hz et 20 000Hz, en dessous et au-delà, il est rare de parvenir à entendre le son.

Pour réaliser une ondulation, vous pouvez utiliser la fonction **cos**, il vous suffira d'aller de 0 à 2 Pi sur le temps d'une unique ondulation pour réaliser celle-ci. L'onde sonore sera une sinusoïde et très douce.

Vous pouvez aussi, plus simplement, mais le son sera moins agréable, simplement remplir la première moitié du temps de l'ondulation par la valeur -25 000, par exemple, et la seconde par 25 000. L'onde sonore sera du carré et plutôt sèche.

Voici quelques fréquences que vous pouvez essayer :

Do Ré Mi Fa Sol La Si Do : 262, 294, 330, 349, 392, 440, 494, 524

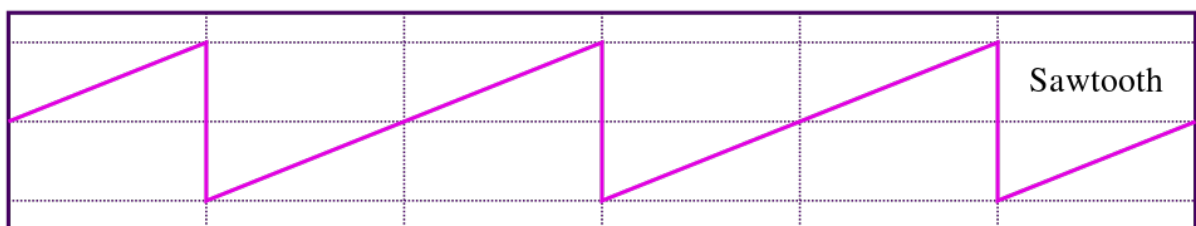
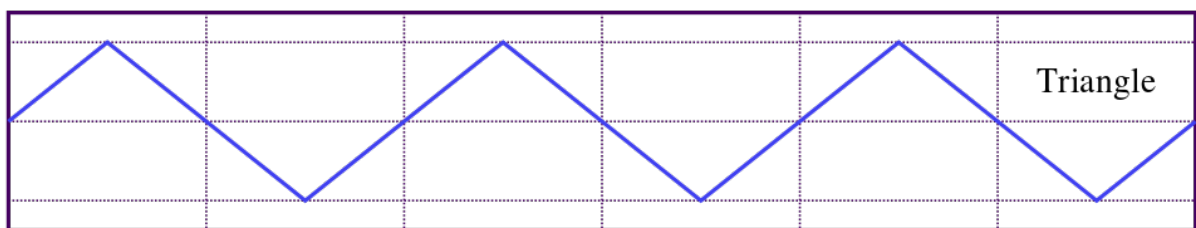
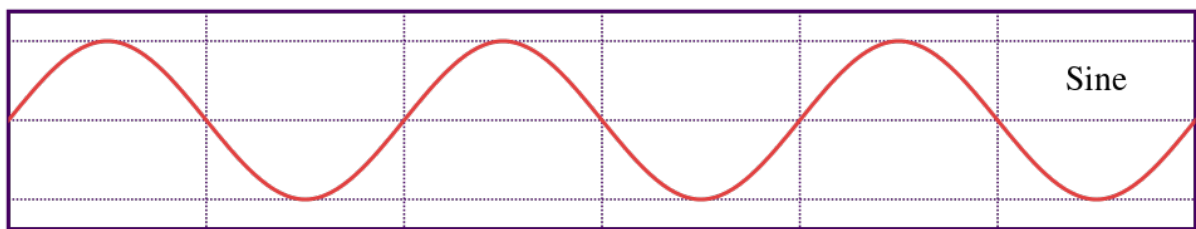
Vous remarquerez que le do plus aigu a le double de fréquence du do grave : pour passer d'un octave à l'autre, il suffit de multiplier par 2 la fréquence d'une note.





## 06 – Des ondes diverses

Il est possible de générer des sons très différents simplement en définissant des formes d'ondes diverses. Si vous êtes d'attaque, écrivez une fonction pour chaque ! Voici quelques ondes habituelles :



La sinusoïde est celle générée par **cos** ou **sin**. L'onde carrée est celle qui est générée en établissant deux valeurs stables sur différentes parties de l'onde. L'onde triangulaire définit deux phases : une montante linéaire et une descendante linéaire. L'onde dent de scie définit une unique phase montante faisant toute l'ondulation – une onde seulement descendante est bien sûr possible aussi. Il est possible d'effectuer d'autres transformations, nous les verrons ultérieurement.