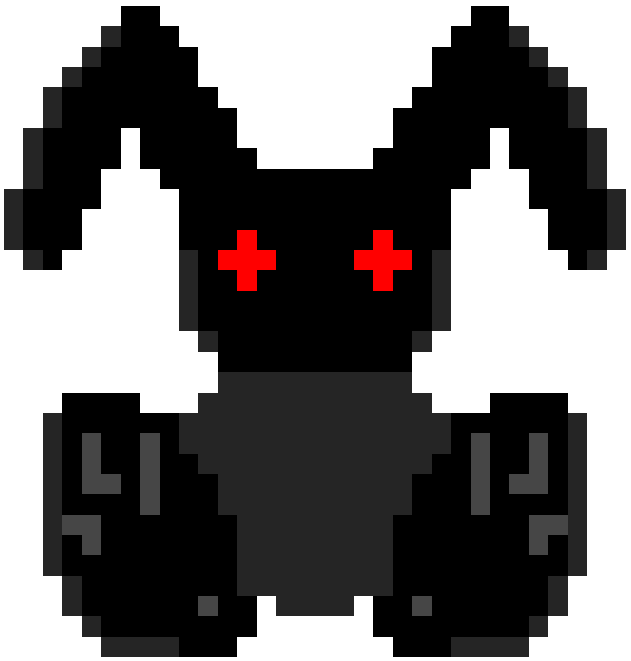

LAPINS NOIRS

- La Caverne Aux Lapins Noirs -

MELANGEUR

1+1=1

- La Caverne aux Lapins Noirs -

INDEX

01 – Avant-propos

Votre travail doit être rendu via le dossier **~/tp/mixer/.**

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 seul. 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.

03 – 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.

Si vous ne l'avez pas fait, 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.

04 – Re-échantillonner

Vous allez charger un fichier son à l'aide de la fonction **bunny_load_effect**. Quelques différences visibles vont apparaître : la fréquence d'échantillonnage ne sera probablement pas 44100Hz ! A la place, elle sera ce que le fichier utilisait comme fréquence, tout simplement…

Écrivez la fonction suivante :

```
t_bunny_effect *std_resample(t_bunny_effect       *fx);
```

La fonction **std_resample** prend en paramètre un son quelconque et en crée un neuf avec une fréquence d'échantillonnage de 44100Hz.

Son rôle est de transférer toutes les données situés dans **fx** en adaptant celle-ci à la fréquence du son créé, sans déformation. Si vous vous contenter de copier les informations de **fx** dans le nouveau son et que sa fréquence était inférieure à celle du nouveau son, le son sera accéléré (et donc plus aigu), sinon il sera ralenti (et donc plus grave) **Vous devez conserver la nature du son, et donc adapter la donnée à la nouvelle fréquence.**

Une bonne première étape est d'abord de chercher la longueur du son en terme de sample… Ensuite, à vous de parcourir *le nouveau son, sample par sample,* et de trouver quel sample dans l'ancien son correspond.

L'ancien son n'est pas détruit par **std_resample**.

05 – Mélange

Écrivez la fonction suivante :

```
t_bunny_effect *std_mix(int                nbr,
                        ...);
```

La fonction std_mix crée un nouveau son à partir des nbr sons passé en paramètres. Tous ces sons doivent déjà avoir été ré-échantillonné à 44100Hz. Le nouveau son est la somme de tous ces sons.

Pourquoi ne pas sauvegarder dans un programme le résultat à l'aide de bunny_save_effect ?