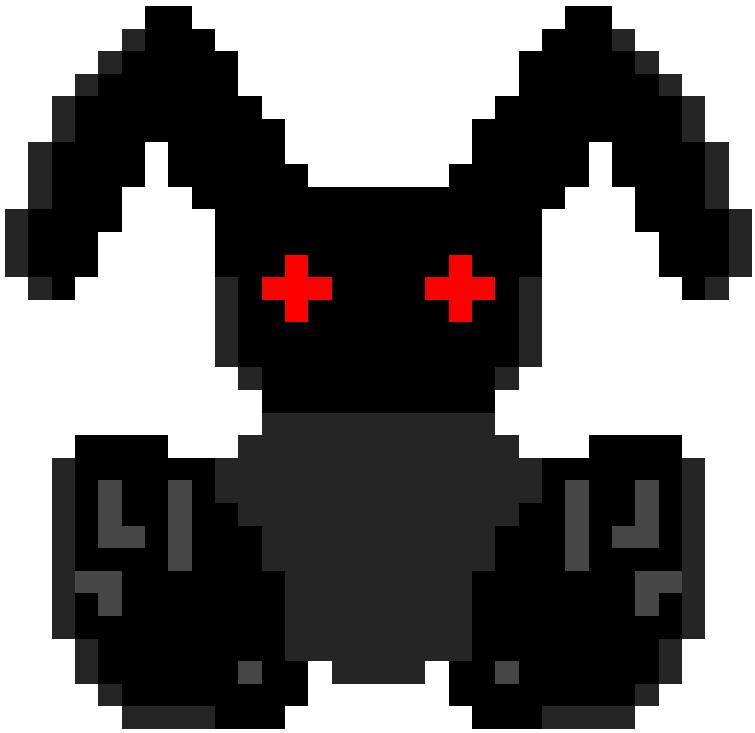

LAPINS NOIRS

- Le Laboratoire aux Lapins Noirs -

Ce mini-projet consiste à réaliser un logiciel d'affichage d'image au format vectoriel simple.

INDEX

01 – Avant-propos

Votre travail doit être rendu via le dossier **~/projets/vectrex/** 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 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 – 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 – VECTREX ?

La « VECTREX » est une console de jeu apparue en 1982. Elle a été conçue par Jay Smith et a la particularité d'être une console à affichage vectoriel.

Qu'est ce qu'un affichage vectoriel ? Une console plus habituelle dispose d'une mémoire ou chaque cellule contient la couleur d'un bloc de pixel ou d'un pixel seul. Cela signifie que pour faire apparaître une forme à l'écran, il faut écrire dans cette fameuse mémoire qui sera ensuite parcourue et affichée par l'écran.

Une console vectorielle ne dispose pas d'une mémoire utilisé comme tel. A la place, elle dispose d'une mémoire contenant des ordres de dessin. Par exemple : dessiner une ligne du point A au point B. Ainsi, au lieu d'avoir a remplir sa surface d'affichage a 100 % de couleurs déterminés, une console vectorielle demande a l'écran de tracer certains formes spécifiques.

Les opérations de remplissage de la mémoire dans une console classique étant extrêmement coûteuse d'un point de vue performance, la possibilité de s'en passer permettait aux développeurs de jeux des originalités comme la 3D.

Cette machine était tout à fait modeste si on la compare aux standards d'aujourd'hui : son microprocesseur était cadencé à 1,6 MHz et elle n'était doté que de 1Ko de RAM. C'est sur cette console que le premier casque de réalité virtuelle, conçu par John Ross, est apparu.

05 – Nature du projet

Le projet VECTREX consiste en la réalisation d'un afficheur de fichier d'un format particulier. Voici comme il s'utilise :

```
$> ./vectrex -h
Usage is :
     ./vectrex width height files+ -o output_file
```

Le programme vectrex prend en paramètre la taille de la fenêtre à ouvrir, ainsi qu'un ou plus fichiers de configuration décrivant des formes à dessiner dans celle-ci.

Voici deux exemples de fichiers. Le premier est un format simple INI.

```
$> cat vectrex.ini
Shape="Dot"
Color=255, 0, 0
Coordinates=
10, 10,
20, 20
```

Ce fichier décrit une forme de type « Points ». Cela signifie que « Coordinates » contiendra des couples d'entiers, représentant des coordonnées X et Y qu'il faudra afficher. Les points seront de la couleur déterminé par « Color ». Le format de la couleur est « Color = Rouge, Vert, Bleu ».

Vous n'avez pas à effectuer le découpage du fichier de configuration vous-même dans ce projet- : Vous pouvez utiliser les fonctions des modules « bunny_ini » et « bunny_configuration » de la LibLapin.

Ci-dessous, un autre type de fichier au format plus complexe, DABSIC :

```
$> cat vectrex.dab
{Shapes
  [
    Type = "Dot"
    {Coordinates
      [
        Position = 10, 10
        Color = 255, 0, 0
      ],
      [
        Position = 30, 30
        Color = 255, 0, 0
      ]
    }
  ]
}

$>
```

Le champ « Shapes » est un tableau contenant ici une unique case. Cette première case contient un champ « Type » contenant la valeur « Dot », de la même manière qu'en INI. Un autre tableau, « Coordinates » est ensuite présent, contenant ici deux cases. Chacune de ces cases représentant un point. Les champs « Position » et « Color » donnent des informations concernant ces points.

Le champ « Shapes[].Coordinates[].Color » est optionnel si un champ « Shapes[].Color » est spécifié.

Le support du format INI par votre programme est obligatoire.
Le support du format DABSIC est recommandé.

06 – Lecture et affichage de points

Votre premier travail va consister à parvenir à ouvrir une fenêtre à la bonne taille, ainsi qu'à charger un fichier de configuration et à l'afficher.

Les coordonnées indiquées dans le fichier de configuration sont indiqués par rapport au centre de la fenêtre et non par rapport au coin supérieur gauche !

Votre programme doit comporter la fonction suivante, qui sera évaluée séparément du logiciel vectrex lui-même :

```
void std_vectrex_pixel(t_bunny_pixelarray    *px,
                       t_bunny_position      pos,
                       unsigned int          color)
```

Cette fonction dessinera dans **px**, à la position **pos** (par rapport au centre de px), un pixel de couleur **color**.

07 – Tracé de lignes

Votre second travail consistera à gérer un nouveau type de forme. Cette forme est **« Line »** et représente une ligne.

Lorsqu'un fichier emploi la forme ligne, les listes de coordonnées, au lieu de comporter seulement un couple X/Y en comporteront deux. Si un ensemble de coordonnées n'est pas complet (le nombre de couple est impair), l'ensemble incomplet est ignoré.

Votre programme doit comporter la fonction suivante, qui sera évaluée séparément du logiciel vectrex lui-même :

```
void std_vectrex_line(t_bunny_pixelarray     *px,
                      t_bunny_position       *pos,
                      unsigned int           color)
```

Cette fonction dessinera dans **px** une ligne, depuis la position **pos[0]** (par rapport au centre de px) jusqu'à **pos[1]** (idem), de couleur **color**.

08 – Courbes

Cette partie est optionnelle. Vous allez à nouveau gérer un nouveau type de forme. Cette forme est **« Curve »** et représente une ligne courbe.

Lorsqu'un fichier emploi la forme courbe, les listes de coordonnées, au lieu de comporter seulement un couple X/Y en comportent 4. Si un ensemble de coordonnées n'est pas complet, l'ensemble incomplet est ignoré.

Votre programme doit comporter la fonction suivante, qui sera évaluée séparément du logiciel vectrex lui-même :

```
void std_vectrex_curve(t_bunny_pixelarray    *px,
                       t_bunny_position      *pos,
                       unsigned int          color)
```

Cette fonction dessinera dans **px** une courbe, depuis la position **pos[0]** (par rapport au centre de px) jusqu'à **pos[3]** (idem), de couleur **color**. Les position **pos[1]** et **pos[2]** représentant des poids influençant la courbure de la courbe.

09 – Triangles

Cette partie est optionnelle. Vous allez à nouveau gérer un nouveau type de forme. Cette forme est **« Triangle »** et représente un triangle

Lorsqu'un fichier emploi la forme triangle, les listes de coordonnées, au lieu de comporter seulement un couple X/Y en comportent 3. Si un ensemble de coordonnées n'est pas complet, l'ensemble incomplet est ignoré.

Votre programme doit comporter la fonction suivante, qui sera évaluée séparément du logiciel vectrex lui-même :

```
void std_vectrex_triangle(t_bunny_pixelarray      *px,
                          t_bunny_position        *pos,
                          unsigned int            color)
```

Cette fonction dessinera dans **px** un triangle dont les coins sont aux coordonnées **pos[0]**, **pos[1]** et **pos[2]**. L'ensemble du triangle sera rempli par la couleur **color**.

11 – Quadrilatères

Cette partie est optionnelle. Vous allez à nouveau gérer un nouveau type de forme. Cette forme est **« Quad »** et représente un quadrilatère.

Lorsqu'un fichier emploi la forme quadrilatère, les listes de coordonnées, au lieu de comporter seulement un couple X/Y en comportent 4. Si un ensemble de coordonnées n'est pas complet, l'ensemble incomplet est ignoré.

Votre programme doit comporter la fonction suivante, qui sera évaluée séparément du logiciel vectrex lui-même :

```
void std_vectrex_quad(t_bunny_pixelarray          *px,
                      t_bunny_position            *pos,
                      unsigned int                color)
```

Cette fonction dessinera dans **px** un quadrilatère d'après les coordonnées situés dans **pos** et rempli de couleur **color**. Vous pouvez tout à fait utiliser **std_vectrex_triangle** pour vous aider.