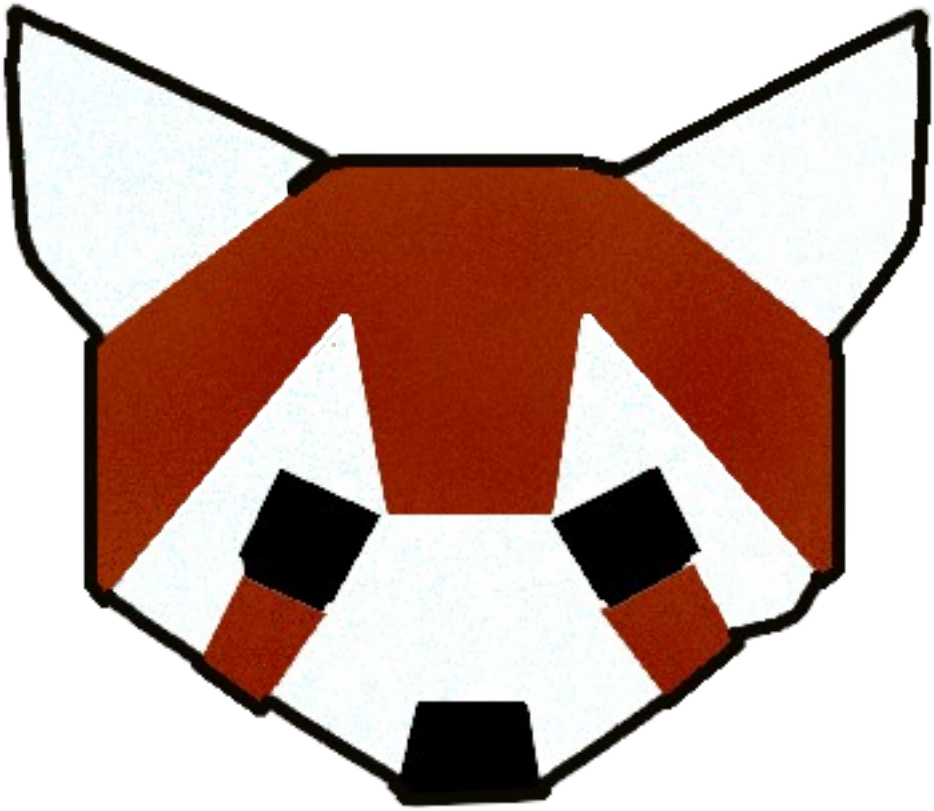
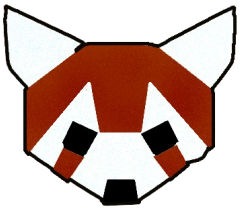


PANDALAB

Transparence

Dessiner un pixel d’une couleur pas tout à fait franche

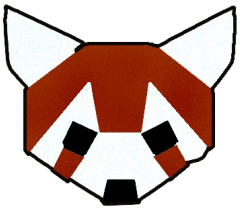
- PandaLab -
pedagogie@ecole-89.com



INDEX

01 – Avant-propos
02 – Fonctions autorisées
03 – Méthode de construction

03 – Poser un pixel avec transparence
04 – Un joli programme



01 – Avant-propos

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

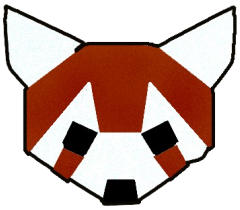
Ce travail est a effectuer seul. De plus il s'agit d'un examen. Vous n'avez donc pas le droit de communiquer avec vos camarades.

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. Nous avons cependant fait le choix de vous interdire son utilisation, 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 à l'exception de celles que nous vous autoriserons explicitement.

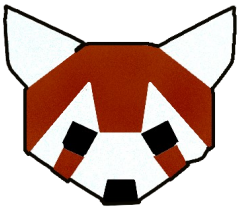
Pour cette activité, issu de la LibC, vous n'avez le droit qu'à la liste suivante:

- 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 laLibLapin, 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.



03 – Méthode de construction

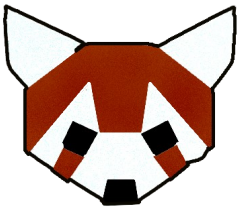
Il peut vous être demandé d'écrire des programmes ou des fonctions.

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 vousavez 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 toujoursla 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 votreprojet rendu.



04 – Poser un pixel avec transparence

ratio.c get_pixel.c color_ratio.c

Réalisez les fonctions suivante comme étape intermédiaire:

```
double std_ratio(int          value,
                 int          ref);
```

La fonction **std_ratio** calcule un coefficient de **value** sur **ref**. C'est à dire que si **value** vaut **ref**, la fonction renvoi 1. Si **value** vaut la moitié de **ref**, la fonction renvoi 0.5, etc.

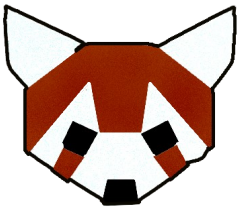
```
unsigned int std_color_ratio(unsigned char       top,
                             unsigned char       bottom,
                             unsigned char       transparency);
```

La fonction **std_color_ratio** calcule le mélange entre **top** et **bottom** d'après **transparency**. Le maximum d'une composante de couleur étant 255 et le minimum 0. Si **transparency** vaut 0, alors 0% de top est utilisé et 100% de **bottom** est utilisé pour le calculde la valeur finale de la composante, mélange des deux. Si **transparency** vaut 255, alors100% de top est utilisé et 0% de **bottom** est utilisé. Si **transparency** vaut 128, alors les deuxsont utilisés à hauteur de 50%.

```
unsigned int std_get_pixel(t_bunny_pixelarray       *px,
                           t_bunny_position         pos);
```

La fonction **std_get_pixel** renvoi la couleur présente à la position **pos** dans **px**.
Vous allez maintenant modifier votre fonction **std_set_pixel** de manière à gérer lacomposante de transparence: à la place de simplement gérer l'ajout d'une couleur à uneposition donnée, vous allez **d'abord calculer cette couleur d'après la transparence, la couleur à poser et la couleur qui se trouvait la avant**. N'hésitez pas à utiliser l'uniont_bunny_color. Sur la page suivante, vous trouverez un programme de test permettant de vérifier lefonctionnement de votre pose de pixel transparent. Certaines fonctions sont nécessaires et vousdevrez évidemment les fournir.

```
void sweet_noise (t_bunny_pixelarray        *px)
{
        t_bunny_position pos;
        t_bunny_color clr;
        int i;
        clr.full = BLACK;
        clr.argb[ALPHA_CMP] = 64;
        // Votre clear_pixelarray DOIT utiliser le set_pixel
        // qui gère la transparence
        e89_clear_pixelarray(px, clr.full);
        i = 0;
        while (i < 200)
        {
                pos.x = rand()%px->clipable.buffer.width;
                pos.y = rand()% px->clipable.buffer.height;
                std_set_pixel(px, pos, WHITE);
                i = i + 1;
        }
}
```



04 – Un joli programme

set_square.c

Ecrivez la fonction suivante :

```
void        std_set_square(t_bunny_pixelarray       *px,
                           t_bunny_area             area,
                           unsigned int             color)
```

Cette fonction dessine un carré de couleur color dans l'espace décrit par area (x, y, w,h) dans px. Arrangez pour que cette fonction utilise votre dessin de pixels transparents…
Votre programme disposera de deux éléments: un **t_bunny_window** qui ne serviraqu'au dessin final et un **t_bunny_pixelarray** dans lequel tout se passera.

Votre programme dessinera des carrés aléatoirement situés dans votre **pixelarray** (sans jamais déborder). La couleur de ces carrés seront aléatoires, de sorte à ce que la transparence et la couleur varie à chaque fois. A chaque tour de boucle, vous remplirez également ce **pixelarray** d'une couleur noire très transparente, de sorte que les carrés dessinés il y a longtemps s'effacent progressivement… Après, vous afficherez votre **pixelarray** dans votre fenêtre.
Faites cela en boucle, idéalement, en utilisant **bunny_set_display_function**,**bunny_set_key_response** et **bunny_loop**. Votre programme quittera par une pression sur latouche d'échappement