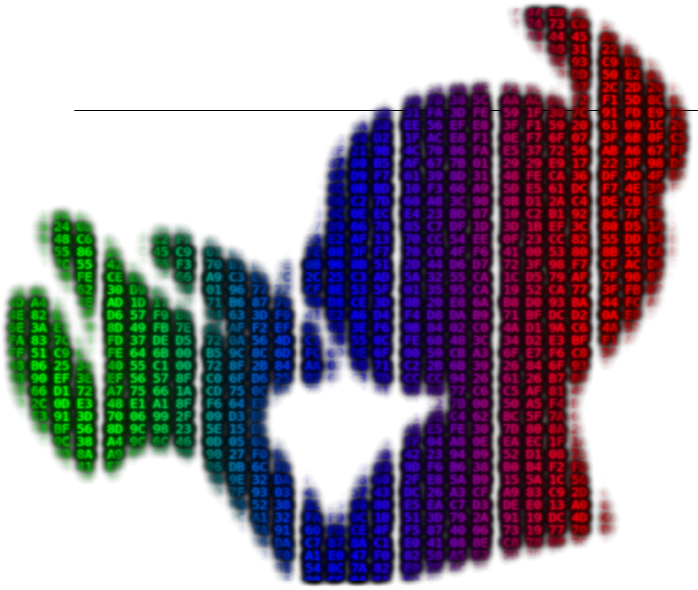


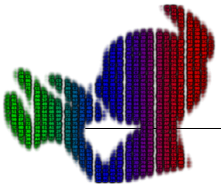
DAEMONLAB

# LSEEK

Lire et écrire a des positions données
Nous ré-utiliserons les fichiers Petite DB.

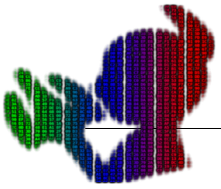
- DaemonLab -

*Ce TP aborde l'utilisation de l'appel système **lseek**.*





# INDEX





# 01 – Détails administratifs

Votre travail doit être rendu via votre bibliothèque, **libstd**.

**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.

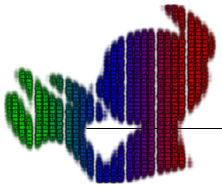
---

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
- time
- lseek
- write
- alloca
- atexit
- rand
- sin
- sqrt
- malloc, free

# 03 – Afficher

Définissez l'élément suivant dans vos fichiers en-têtes. uint32_t fait partie de stdint.h.

```
typedef struct      s_small_db
{
  union
  {
    char            fourcc[4];
    uint32_t        key;
  };
  char              data[28];
}                   t_small_db;
```

Écrivez la fonction suivante :

```
int       std_print_small_db(const t_small_db     *db)
```

Cette fonction va afficher le contenu du tableau de **t_small_db**. Une valeur nulle pour l'attribut **key** signifiera qu'on est arrivé au bout du tableau, de la même manière qu'un caractère '\0' signifie la fin d'une chaîne de caractère.
L'affichage se fera de la façon suivante :

abcd : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

« abcd » représentant les 4 caractères du champ « fourcc ». FourCC signifiant « Four character code », soit code à 4 caractère. Ce champ est censé n'être constitué que de caractères imprimables, vous pouvez donc les afficher sans traitement. **Attention à l'absence de \0 !**

Ces 4 caractères sont suivi d'un symbole ':' et d'un espace avant d'être suivi de 28 valeurs en hexadécimal représentant les 28 octets du tableau **data**. Chaque valeur est séparée par un espace et la ligne est terminée par un saut de ligne.

Le nombre de caractères écrit est renvoyé. En cas d'erreur, une valeur négative ou nulle dont la valeur absolue est le nombre de caractère déjà écrit est renvoyée.

Ci-après, une fonction **main** pour tester votre programme.

```
int             main(void)
{
  t_small_db    db[3] =
  {
    {
      .fourcc = {'c', 'o', 'c', 'o'},
      .data = { 0xFF, 0x00, xFF, 0x00}
    },
    {
      .fourcc = {'L', 'A', 'P', '1'},
      .data = { 'l', 'o', 'l', '\n', 0}
    },
    {
      .key = 0
    }
  };

  std_print_small_db(db);
  return (0);
}
```

Vous devriez voir apparaître deux lignes de texte résumant les valeurs fixées dans ces tableaux. Les champs n'ayant pas été assignés seront mis à zéro par le compilateur, comme c'est le cas lors de ce genre d'initialisation de variables sur une ligne.

```
coco: FF 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LAP1: 6C 6F 6C 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

# 04 – Charger un seul élément

Écrivez la fonction suivante :

```
bool            std_peek_small_db(int           fd,
                                  int           index,
                                  t_small_db    *out)
```

Cette fonction charge le contenu d'un unique **t_small_db** situé dans le fichier ouvert et représenté par **fd**, situé à la position **index** (en taille de **t_small_db** et non en octet) dans l'espace situé dans **out**.

Cette fonction renvoi **vrai** si elle réussi à lire et remplir entièrement la structure passée en paramètre. Sinon elle renvoi **faux**.

Afin de pouvoir réaliser cette opération, vous devrez utiliser la fonction **lseek**. Celle-ci permet de déplacer le point de lecture/écriture à un endroit arbitraire. Cet appel système échouera si il n'est pas utilisé avec un **file descriptor représentant un fichier**. Il ne fonctionnera pas par exemple sur l'entrée standard ni sur une connexion réseau.

Le point de lecture dans le fichier ne doit pas être altéré par cette fonction : cela signifie que vous devez le sauvegarder en début de fonction puis le restaurer en fin de fonction.

```
int             main(void)
{
  int           fd = open("./small.db", O_RDWR);
  t_small_db    db[2] =
  {
    { .key = 0 },
    { .key = 0 }
  };

  std_peek_small_db(fd, 0, &db[1]);
  // Puts
  return (0);
}
```

# 05 – Sauvegarder un seul élément

Écrivez la fonction suivante :

```
bool      std_poke_small_db(int                fd,
                            int                index,
                            const t_small_db *out)
```

Cette fonction écrit le contenu d'un unique **t_small_db** pointé par **out** dans le fichier ouvert et représenté par **fd**, à la position **index** (en taille de **t_small_db** et non en octet).

Cette fonction renvoi **vrai** si elle réussi à écrire entièrement la structure passée en paramètre. Sinon elle renvoi **faux**.

Afin de pouvoir réaliser cette opération, vous devrez utiliser la fonction **lseek**. Celle-ci permet de déplacer le point de lecture/écriture à un endroit arbitraire. Cet appel système échouera si il n'est pas utilisé avec un **file descriptor représentant un fichier**. Il ne fonctionnera pas par exemple sur l'entrée standard ni sur une connexion réseau.

Le point de lecture dans le fichier ne doit pas être altéré par cette fonction : cela signifie que vous devez le sauvegarder en début de fonction puis le restaurer en fin de fonction.

**Si votre fonction écrit après la fin du fichier, vous devez vous assurer le remplissage des structures situés entre la fin de votre fichier et votre point d'écriture par des 0.**