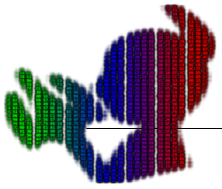
DAEMONLAB

Petite BD 1

« BD » pour « Base de donnée »
Dans les faits, vous allez apprendre à manipuler des fichiers binaires.

- DaemonLab -

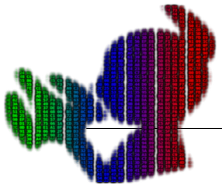
Ce TP aborde la manipulation de fichiers binaires.



INDEX

01 – Avant-propos
02 – Fonctions autorisées

03 – Afficher
04 – Charger
05 – Sauvegarder
06 – Rechercher
07 – Ajouter et retirer
08 – Accès à un type complexe



01 – Avant-propos

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.

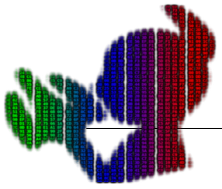
Ce travail est a effectuer seul. Vous pouvez bien sur échanger avec vos camarades, néanmoins vous devez être l'auteur de votre travail. Utiliser le code d'un autre, c'est **tricher**. Et tricher annule **toutes** les médailles que vous avez reçu sur l'activité. La vérification de la triche est réalisée 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
- write
- alloca
- atexit
- rand
- sin
- sqrt
- malloc, free

Pour information, va_start, va_arg et va_end sont des macros, vous y avez donc droit.

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.

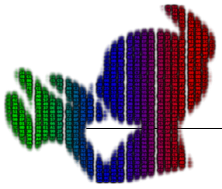
Ci-après, une fonction **main** pour tester votre programme.

```
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

Écrivez la fonction suivante :

```
t_small_db      *std_load_small_db(const char *file)
```

Cette fonction charge le contenu du fichier passé en paramètre et le renvoi sous la forme d'un tableau.

Vous trouverez un fichier au bon format **file.db** avec ce TP que vous pourrez utiliser pour tester votre fonction. Ce fichier contient l'équivalent du tableau de l'exercice précédent.

Vous aurez besoin de **open**, **read**, **close** et **malloc**.
Vous pouvez au choix utiliser **realloc** ou **lseek**.

Cette fonction renvoi **NULL** en cas d'erreur. Si l'erreur est lié à un format invalide dans le fichier chargé, le code **errno** utilisé sera **EINVAL**.

Attention, les fichiers **.db** ne contiennent pas le terminateur : **c'est à vous de l'ajouter.**

05 – Sauvegarder

Écrivez la fonction suivante :

```
int         std_save_small_db(const char  *file,
                              t_small_db  *db)
```

Cette fonction écrit dans le fichier file le contenu du tableau db. Le terminateur n'est pas enregistré !

Cette fonction renvoi le nombre d'octets écrit dans le fichier. Si une erreur se produit, le nombre d'octet écrit multiplié par -1 est renvoyé.

06 – Rechercher

Écrivez les fonctions suivantes, ainsi que la macro :

```
t_small_db    *std_searchk_small_db(t_small_db    *db,
                                    uint32_t      key)

t_small_db    *std_searchc_small_db(t_small_db    *db,
                                    const char    *fcc)

#define        std_search_small_db(db, key)
```

Ces fonctions cherchent dans les tableaux passé en paramètre la case contenant **key** et le renvoi. La première, **searchk** base sa recherche sur le champ **key** de l'union, tandis que la seconde base sa recherche sur le champ **fourcc**.

Si la case n'est pas trouvée ou si une erreur arrive, **NULL** est renvoyé. Si **fcc** est trop long, l'erreur enregistrée dans **errno** sera **EINVAL**. Si **key** traduit en caractères donne des caractères non imprimable, **EINVAL** sera également utilisé.
Si la case n'est simplement pas trouvée, **errno** sera établi à 0.

La macro **std_search_small_db** utilisera le nouveau mot clef du **C11**, à savoir **_Generic**. Sa syntaxe est la suivante :

```
_Generic(type_de_la_donnée,
        type1 : expansion_si_la_donnée_est_de_type1,
        type2 : expansion_si_la_donnée_est_de_type2,
        default : expansion_si_la_donnée_est_de_un_autre_type
)
```

Le champ **default** est optionnel. L'objectif de cette macro est d'appeler soit **searchk** soit **searchc** en fonction du type du paramètre **key** de la macro.

07 – Ajouter et retirer

Écrivez la fonction suivante :

```
bool            std_add_small_db(t_small_db         *db,
                                 const t_small_db   *new)
```

Cette fonction ajoute dans **db** la structure **new**.
Vous pouvez utiliser pour cela la fonction **realloc.**

La fonction renvoi **vrai** si elle réussit, sinon elle renvoi **false.** Si l'erreur empêchant l'ajout de **new** est que son FourCC est déjà présent, alors **errno** sera établi à **EINVAL**.

Écrivez les fonctions suivante ainsi que la macro :

```
bool        std_removek_small_db(t_small_db   *db,
                                 uint32_t     key)

bool        std_removec_small_db(t_small_db   *db,
                                 const char   *fcc)

#define     std_remove_small_db(db, key)
```

Ces fonctions et macros retirent la case dont la clef ou le code est **key** ou **fcc**. Vous pouvez utiliser **realloc** pour changer la taille de votre tableau. Évidemment, si l'on retire une case en plein au milieu de **db**, vous devez vous arranger pour qu'aucun trou ne perdure.

08 – Accès à un type complexe

Écrivez les macros suivante :

```
#define   std_catch_small_db(db, type)
#define   std_get_small_db(db, key, type)
```

La première macro renvoi un **type*** pointant sur la première case du champ **data** situé dans le **t_small_db** passé en paramètre.

La seconde macro prend en paramètre un tableau entier, recherche dedans l'élément dont la clef est **key** et effectue un renvoi du même type que **catch_small_db** sur l'élément trouvé.

Ces macros peuvent bien sur appeler des fonctions.

Étant donné que la taille du champ **data** n'est que de 28 octets, évidemment, pour être cohérent, il faudrait que **type** fasse 28 octets maximum, sans quoi des erreurs sont à prévoir.

Ci après, un exemple d'utilisation.

```
typedef struct      s_sound_wave
{
  int32_t           start;
  int32_t           stop;
  int32_t           instrument;
  int32_t           frequency;
}                   t_sound_wave;

int             main(void)
{
  t_small_db    db[2] = {0}; // Tout est à 0.
  t_sound_wave  sw;
  t_sound_wave  *ptr;

  memcpy(&db.fourcc[0], "midi", 4);
  sw.start = 0;
  sw.stop = 20000;
  sw.instrument = 3;
  sw.frequency = 220;
  memcpy(&db.data[0], sw, sizeof(sw));
  ptr = std_get_small_db(&db, "midi", t_sound_wave*);
  // ptr pointe sur &db.data[0] mais se lit comme
  // un t_sound_wave*
}
```