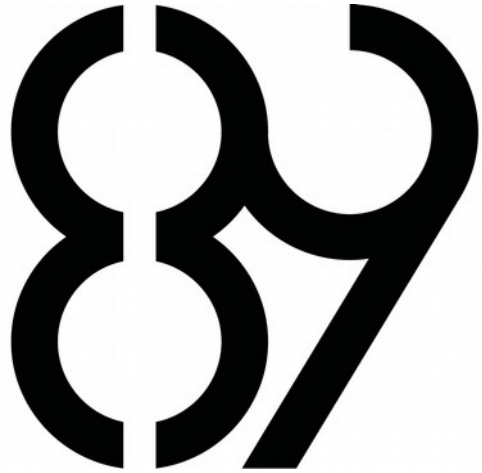
D A E M O N L A B



Petite BD II

« BD » pour « Base de donnée »

- DaemonLab -
pedagogie@ecole-89.com

INDEX

01 – Détails administratifs

Votre travail doit être envoyé via l'interface de ramassage de l'Infosphère :

Pour cette activité, vous rendrez votre travail sous la forme d'une archive au format tar.gz. Cette archive devra contenir l'ensemble de votre travail tel que demandé dans la section 4.

Pour créer cette archive .tar.gz, il vous suffit d'utilliser la commande suivante :

```
$> tar cvfz mon_fichier.tar.gz fichier1 fichier2 fichier3
```

Le nom « mon_fichier.tar.gz » étant à remplacé par le nom que vous souhaitez donner votre fichier archive, et « fichier1 », « fichier2 », « fichier3 » par les fichiers ou dossiers que vous souhaitez mettre dans cette archive. Vous pouvez vérifier le contenu de votre archive à l'aide de la commande « **`tar -t mon_archive.tar.gz`** ».

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.

Médailles accessibles :

Réussir à rendre :
Vous avez réussi à envoyer votre travail au système de correction.

02 – Propreté de votre rendu

Votre rendu, c'est à dire le contenu de l'archive ou du dépôt que vous entrez sur l'interface du TechnoCentre, doit respecter **strictement** l'ensemble des règles suivantes :

- Il ne doit contenir **aucun** fichier objet. (*.o)
- Il ne doit contenir **aucun** ichier tampon. (*~, #*#)
- Il ne doit pas contenir votre production finale (programme ou bibliothèque)

La présence d'un fichier interdit mettra immédiatement fin à votre évaluation.

Médailles accessibles :

Rendu propre
Votre rendu respecte les règles de propretés imposées.

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       e89_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

« abcd » représentant les 4 caractères du champ « fourcc ». FourCC signifiant « Four character code », soit code à 4 caractère. Seuls les caractères imprimables sont supposés valide.

Ces 4 caractères sont suivi d'un symbole ':' et d'un espace avant d'être suivi de 28 valeurs en hexadecimal 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.

```c
int             main(void)
{
  t_small_db    db[3] =
  {
    {
      .fourcc = {'a', 'b', 'c', 'd'},
      .data = { 'e', 'f', 'g', 'h', 0}
    },
    {
      .fourcc = {'A', 'B', 'C', 'D'},
      .data = { 'E', 'F', 'G', 'H', 0}
    },
    {
      .key = 0
    }
  };

  e89_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 à 0.

04 – Charger

Écrivez la fonction suivante :

```
t_small_db      *e89_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         e89_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    *e89_searchk_small_db(t_small_db    *db,
                                    uint32_t      key)

t_small_db    *e89_searchc_small_db(t_small_db    *db,
                                    const char    *fcc)

#define        e89_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 **e89_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            e89_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        e89_removek_small_db(t_small_db   *db,
                                 uint32_t     key)

bool        e89_removec_small_db(t_small_db   *db,
                                 const char   *fcc)

#define     e89_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 millieu de db, vous devez vous arranger pour qu'aucun trou ne perdure.

08 – Accès à un type complexe

Écrivez les macros suivante :

```
#define   e89_catch_small_db(db, type)
#define   e89_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 sur l'élément trouvé.

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;
}                   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;
  memcpy(&db.data[0], sw, sizeof(sw));
  ptr = e89_get_small_db(&db, "midi", t_sound_wave*);
  // ptr pointe sur &db.data[0] mais se lit comme
  // un t_sound_wave*
}
```


09 – Gestion d'un fichier passablement *incorrect*

Vous trouverez joint avec ce TP des fichiers présentant des défaillances spécifiques :

- Le fichier **duplicate.dbx** contient plusieurs fourcc identique. S'agit il d'une erreur ou d'une tentative de disposer de plages de données plus longue par un de vos collègues ? Nous allons partir du principe qu'il s'agit de la seconde hypothèse… Se pourrait-il que ce 'x' en fin de fichier signifie **extended** ?

```
void        *e89_searchk_small_dbx(t_small_db   *db,
                                   uint32_t     key,
                                   size_t       *len)

void        *e89_searchc_small_dbx(t_small_db   *db,
                                   const char   *fcc,
                                   size_t       *len)

#define      e89_search_small_dbx(db, key, len)
```

Contrairement aux fonctions que vous aviez écrit précédemment, ces fonctions et macros ne renvoient pas directement des **t_small_db** mais des données qui sont **allouées** par ces fonctions. Cela signifie que ces données devront être libérées par un appel à **free.**

Ces fonctions et macros écriront également la taille de la donnée dans ***len**. Si **len** est **NULL**, les fonctions renverront **NULL, de même si les clefs ne sont pas trouvées.**

- Le fichier **earlyend.db** contient plusieurs données, puis un fourcc nul, puis d'autres données. Modifiez votre fonction **e89_load_small_db** de sorte à ignorer ce fourcc mais à ne pas vous y arrêter. Continuez jusqu'à la **fin du fichier**.