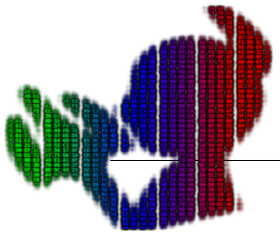
DAEMONLAB

CSV

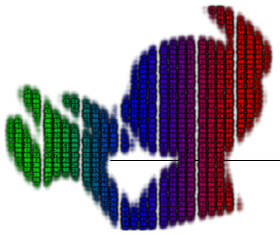
« Comma separated values »

- DaemonLab -

Ce TP aborde les fichiers formatés en suivant un dialecte du style CSV.



INDEX



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.

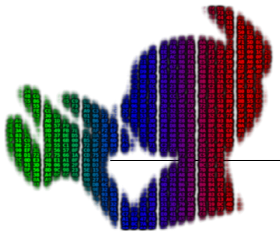
Ce travail est a effectuer seul. Vous pouvez bien sur échanger avec vos camarades, néanmoins vous devez être l'auteur de votre travail. Utiliser le code d'un autre, c'est **tricher**. Et tricher annule **toutes** les médailles que vous avez reçu sur l'activité. La vérification de la triche est réalisée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

Écrivez la fonction suivante :

```
int             std_read_cchar(const char       *str,
                               char             *c)
```

Cette fonction lit dans la chaîne de caractère str le premier caractère qui s'y trouvent et en interprète le symbole. Le caractère est stocké dans *c. La fonction renverra le nombre de caractères lu dans str. Par exemple, si le caractère A majuscule est lu, la fonction renverra 1. Si les caractères backslash \ et n sont trouvés pour former un saut de ligne, la fonction renverra 2.

Ci-dessous, un programme de test :

```
int                 main(void)
{
      const char    *s = "a\\nb";
      char          out[4];
      int           i;
      int           j;

      i = 0;
      j = 0;
      i += std_read_char(s, &out[j++]);
      i += std_read_char(s, &out[j++]);
      i += std_read_char(s, &out[j++]);
      out[j] = '\0';
      i += std_read_char(s, &out[j++]);
      if (i == 4 && out[0] == 'a' && out[1] == '\n'
            && out[2] == 'b' && out[3] == '\0')
            tc_putchar('y');
      else
            tc_putchar('n');
      return (EXIT_SUCCESS);
}
```

04 – C-String

Écrivez la fonction suivante :

```
char            *std_read_cstring(const char   *str)
```

Cette fonction lit dans la chaîne de caractère **str** les caractères qui s'y trouvent et en interprète les symboles afin d'en retourner une copie où ceux là ont été interprété.

Par exemple, la chaîne suivante :
« Coucou\n »
Soit, en hexadecimal 0x43 0x6F 0x75 0x63 0x6F 0x75 0x5C 0x6E 0x00

Doit être transformé en :
0x43 0x6F 0x75 0x63 0x6F 0x75 0x0A 0x00

Soit une transformation de symboles '\' **antislash** et 'n' **lettre n** en '\n' **nouvelle ligne**.

La liste des symboles à transformer est la suivante :
'\a', '\b', '\t', '\n', '\v', '\f', '\r'

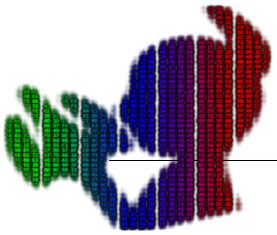
Ainsi que :
'\0xxx' ou xxx est un nombre en **octal**.

Et :
'\xXX' ou XX est un nombre en **hexadecimal**.

Écrivez la fonction suivante :

```
char            *std_write_cstring(const char  *str)
```

Cette fonction effectue l'opération inverse. Elle transforme 0x0A en 0x5C 0x6E par exemple.



05 – CSV

Écrivez la fonction suivante :

```
typedef struct s_csv
{
	int		width;
	int		height;
	char		*data;
}			t_csv;

t_csv			*std_read_csv(const char	*str)
```

Les données d'un fichier CSV sont organisées en deux dimensions, à la manière des données dans un tableur. Le *format CSV est d'ailleurs un format qui est souvent disponible à l'export dans ces logiciels.*

Le déplacement sur la dimension Y est incarnée par des sauts de ligne.

Le déplacement sur la dimension X est incarnée par le point virgule ';'.

Les données que l'on peut trouver à chaque emplacement dans un fichier CSV peuvent être de différentes nature. Dans notre cas, les données seront **toujours** des C-Strings.

Vous écrivez par la suite les fonctions suivantes, se basant sur **std_read_csv**.

```
t_csv			*std_load_csvd(int		fd)
t_csv			*std_load_csv(const char	*file)
```

La fonction **std_load_csv** exploitant bien sur **std_load_csvd**.

De la même façon :

```
char			*std_write_csv(t_csv		*data)
bool			std_save_csvd(t_csv		*data,
					int		fd)
bool			std_save_csv(t_csv		*data,
					const char	*file)
```