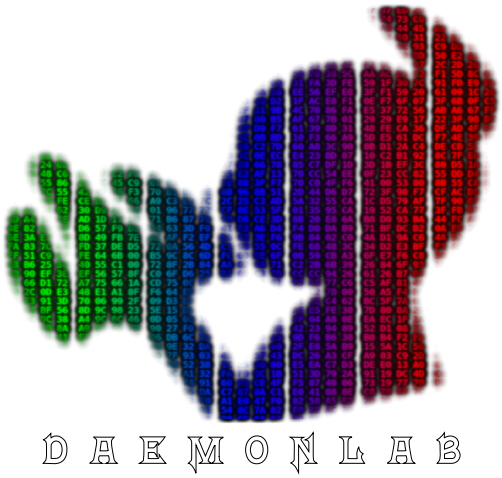

Journée 05

Macro, en-tête, analyse grammaticale

- DaemonLab -
daemonlab@ecole-89.com

Durant cette journée, vous découvrirez un nouvel aspect du C : les en-têtes de fichier. En ouvrant le sujet des en-têtes, nous aborderons aussi les macros et les prototypes.

Nom de code : !pac1j05
Clôture du ramassage : 28/11/2019 23:59

Médailles accessibles :

Définition des médailles à venir

Ce document est strictement personnel et ne doit en aucun cas être diffusé.

INDEX

Avant-propos :

Exercices principaux :

01 – Détails administratifs

Votre travail doit être envoyé via l'interface de ramassage du **TechnoCentre** :

http://technocentre.ecole89.com/ramassage

Le numéro de code présent sur ce sujet vous est propre : vous devrez le renseigner en rendant votre travail. En cas d'erreur, votre travail ne sera pas associée à l'activité et votre travail ne sera pas ramassé.

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 – Règlement quant à la rédaction du code C

En général, le code source de votre programme doit impérativement respecter un ensemble de règles de mise en page définie par les Table de la Norme.

Pour cette activité, nous vous libérons de cette contrainte qui vous sera néanmoins très bientôt imposée pour l'ensemble de vos programmes écrits en C.

04 – Construction partielle de votre rendu

Le programme de correction va construire une sous-partie déterminée de votre rendu afin d'effectuer des tests dessus. En voici les paramètres :

- Les fichiers qui seront compilés sont ceux qui auront l'extension *.c.
- Seuls les fichiers dans le(s) dossier(s) ./ seront compilés.
- Les fichiers seront compilés sans règles de compilation particulière.

- Tous les fichiers seront compilés **ensemble**, cela signifie donc que chaque fonction doit être unique, et que vous pouvez utiliser les fonctions des autres exercices dans chaque exercice.

L'ensemble du code que vous rendez doit pouvoir être compilé.
En cas d'échec de la compilation, vous ne serez pas évalué.

Médailles accessibles :

Construction partielle
Les éléments requis de votre projet se construisent séparement.

05 – Fonctions interdites

Vous n'avez le droit à aucune autre fonction que celle précisée dans la liste ci-dessous :

- write
- malloc
- free

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.

06 – Macro ABS

Mettre dans le fichier : macro.h

Programmez la macro **ABS**.

```
#define ABS(a)     ...
```

La macro **ABS** renvoi la valeur absolue de **a**.

07 – Macro MIN

Mettre dans le fichier : macro.h

Programmez la macro **MIN**.

```
#define MIN(a, b)     ...
```

La macro **MIN** renvoi la plus petite valeur entre **a** et **b**.

08 – Macro MAX

Mettre dans le fichier : macro.h

Programmez la macro **MAX**.

```
#define MAX(a, b)     ...
```

La macro **MAX** renvoi la plus grande valeur entre **a** et **b**.

09 – Macro CLAMP

Mettre dans le fichier : macro.h

Programmez la macro **CLAMP**.

```
#define CLAMP(a, b, c)     ...
```

La macro **CLAMP** renvoi :
- **b** si **a** est plus petit que **b**
- **c** si **a** est plus grand que **c**
- sinon **a**

10 – Macro RCLAMP

Mettre dans le fichier : macro.h

Programmez la macro **RCLAMP**.

```
#define RCLAMP(a, b, c)     ...
```

La macro **RCLAMP** renvoi :
- **c** si **a** est plus petit que **b**
- **b** si **a** est plus grand que **c**
- sinon **a**

11 – Macro DIFF

Mettre dans le fichier : macro.h

Programmez la macro **DIFF**.

```
#define DIFF(a, b)     ...
```

La macro **DIFF** renvoit la valeur absolue de la différence entre **a** et **b**.

12 – Macro ARRAY_LENGTH

Mettre dans le fichier : macro.h

Programmez la macro **ARRAY_LENGTH**.

```
#define ARRAY_LENGTH(a)     ...
```

La macro **ARRAY_LENGTH** renvoit la longueur en case du tableau **a**.

13 – print_split

Fichier : print_split.c

Programmez la fonction **print_split**.

```
int print_split(const char **wordtab);
```

La fonction **print_split** affiche toutes les chaînes de caractères dans **wordtab**. Chaque chaîne est suivie d'un saut de ligne. Le paramètre **wordtab** est un tableau de chaînés de caractères terminé par le pointeur **NULL**. La fonction renvoi le nombre de caractères écrits.

Ci-dessous, un exemple de main de test pour vous aider à générer le fameux **wordtab**.

```
int main(void)
{
	const char *wt[4];

	wt[0] = "abc";
	wt[1] = "def";
	wt[2] = "ghi";
	wt[3] = NULL;
	print_split(wt);
	return (0);
}
```

```
$> gcc print_split.c
$> ./a.out
abc
def
ghi
$>
```


14 – split

Fichier : split.c

Programmez la fonction **split** et son test.

```
char **split(const char *str, char token);
```

La fonction **split** éclate la chaine **str** en fonction du caractère token. Par exemple, si **token** vaut ';' et que **str** vaut "abc;def;ghi", alors le tableau renvoyé par split sera ainsi :

- Sa case 0 contiendra "abc"
- Sa case 1 contiendra "def"
- Sa case 2 contiendra "ghi"
- Sa case 3 contiendra NULL

Si une erreur advient, la fonction renverra **NULL**.

14 – free_split

Fichier : free_split.c

Programmez la fonction **free_split** et son test.

```
void free_split(const char **wt);
```

La fonction **free_split** libère l'ensemble envoyé situé dans wt.

14 – merge

Fichier : merge.c

Programmez la fonction **merge** et son test.

```
char *merge(const char **wt, char token);
```

La fonction **merge** génère une nouvelle chaîne de caractère contenant l'ensemble des chaînes situés dans **wt**, relié par le symbole situé dans **token**. Si **token** vaut '\0', aucun caractère n'est ajouté entre les chaînes.

14 – reverse_split

Fichier : reverse_split.c

Programmez la fonction **reverse_split** et son test..

```
void reverse_split(const char **wt);
```

La fonction **reverse_split** inverse l'ordre des éléments dans **wt**. Si l'on appelle **print_split** après modification, les chaînes seront affichées dans l'ordre inverse. (Les caractères dans les chaînes restent identique!)

15 – strtok_r

Fichier : strtok_r.c

Programmez la fonction strtok_r ainsi que son test:

```
char *strtok_r(char *str, const char *d, char **s);
```