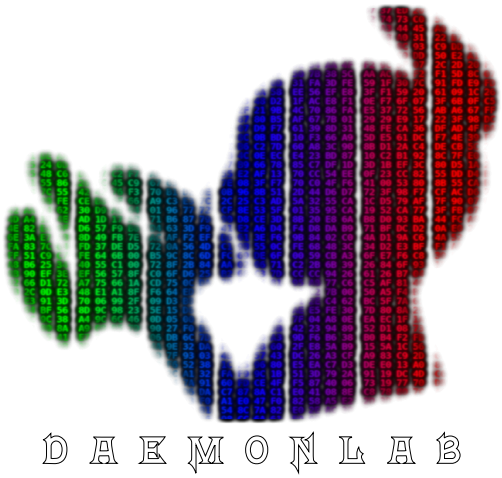

Journée 06

Analyse grammaticale, structures

- DaemonLab -
daemonlab@ecole-89.com

Durant cette journée, vous découvrirez un large panel d'éléments du langage C et progresserez également en analyse grammaticale. ***Attention, la compilation est plus rigide désormais !***

Nom de code : !pac1j06
Clôture du ramassage : 01/12/2019 23:59

Médailles accessibles :

Définition des médailles à venir

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 avec **-W -Wall -Werror**.

- 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 – Entiers formatés

Mettre dans le fichier : intf.h

Définissez dans votre fichier en-tête l'énumération **e_format**, doté d'un alias (« typedefé ») **t_format**. Ses constantes symboles seront dans l'ordre, BIN, DEC, OCT, HEX et END_FORMAT.

Définissez dans ce même fichier la structure **s_intf**, comprenant comme premier attribut un t_format **format** et comme second attribut un entier **val**.

Enfin, implémentez les fonctions suivantes :

```
int print_intf(const t_intf *intf);
t_intf *read_intf(const char *str);
```

La fonction **print_intf** affiche la valeur de **val** dans **intf**, en binaire si **intf→format**, vaut BIN. En décimal si DEC, en octal is OCT, en hexadecimal is HEX et n'affiche rien si END_FORMAT.

La fonction print_intf renvoit le nombre de caractères qui ont été écrit.

07 – Typage dynamique

Mettre dans le fichier : data.h

Définissez dans votre fichier l'énumération **e_type**, comprenant les symboles INT, STR, CHR et END_TYPE. Votre type énuméré disposera d'un alias **t_type**. END_TYPE est un terminateur.

Définissez dans votre fichier l'union **u_data_container**, comprenant les attributs int **integer**, char* **string** et char **character**. Votre type d'union disposera d'un alias **t_data_container**.

Définissez dans votre fichier la structure s_data, comprenant les attributs t_type **type** et t_data_container **value**. Votre type de structure disposera d'un alias **t_data**.

Définissez dans votre fichier le type pointeur sur fonction **t_print_data** définissant des fonctions renvoyant un entier et prenant en paramètre un **pointeur constant** sur **t_data_container**. Le rôle de ce type est d'afficher ses paramètres et de renvoyer le nombre de caractères écrits.

Définissez le tableau **gl_print_data**, qui fera la taille du nombre de types que l'énumération **t_type** peut représenter. Pour chacune de ses cases, vous assignerez une fonction qui sera a même d'afficher le type associé. Par exemple, à la case **STR**, vous assignerez une fonction capable d'afficher une chaîne de caractère à partir de **t_data_container**.

Ces fonctions, vous devez évidemment les écrire, et elles devront être compatible avec le pointeur sur fonction **t_print_data**. C'est à dire prendre les mêmes arguments et renvoyer le même type de valeur. De plus, vous devrez programmer :

```
int print_data(const t_data *data);
t_data *read_data(const char *str);
```

La fonction **print_data** affiche l'union situé dans **t_data** en s'aidant du tableau de pointeur sur fonction **gl_print_data** et de l'attribut **type** de **t_data**.

La fonction **read_data** lit dans **str** et déduit de ce qu'elle lit les paramètres de construction d'un **t_data**. Si **str** contient des chiffres au départ : c'est un entier. Si c'est un seul caractère non chiffre : c'est un caractère. Sinon c'est une chaîne de caractère.

08 – Macro OFFSET_OF

Fichier : offset_of.h

Programmez la macro OFFSET_OF :

```
#define OFFSET_OF(type, attribute)     ...
```

La macro OFFSET_OF prend en paramètre un type de structure ainsi qu'un attribut et renvoi la position dans la structure type de l'attribut. Cette position est en octet.

09 – swab

Fichier : swab.c

Écrivez la fonction swab, ainsi que son test, tel qu'elle est décrit dans son manuel.

10 – memswap

Fichier : memswap.c

Écrivez la fonction memswap dont le prototype est le suivant. N'oubliez pas le test.

```
void memswap(void *a, void *b, size_t len);
```

La fonction memswap intervertir tous les octets de a et de b sur len octets.

11 – play_sounds

Fichier : play_sounds.c

Écrivez la fonction **play_sounds** dont le prototype est le suivant.

```
void boing(void);
void boum(void);
void tchak(void);
void tsong(void);
void play_sounds(int *music);
```

La fonction **play_sounds** utilise un tableau de pointeur sur fonction comprenant à sa case 0, **boing**, à sa case 1, **boum**, à sa case 2, **tchak** et à sa case 3 **tsong**. Le paramètre **music** est un tableau d'entier terminé par -1. Pour chaque valeur valide dans **music**, (c'est à dire correspondant à une case du tableau de pointeur sur fonction), on appelle la fonction associée.

Chacune des fonctions de « son » affiche aussi un saut de ligne après son son.