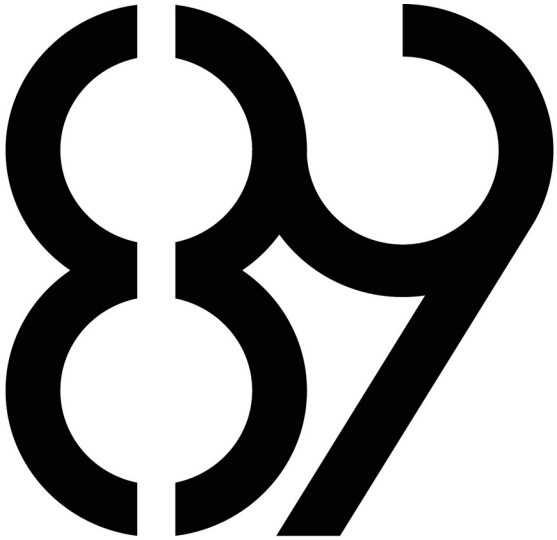

Journée 00

Variables, conditions, boucles

- DaemonLab -
daemonlab@ecole-89.com

Durant cette journée, vous allez pour la plupart d'entre vous programmer pour la première fois. Nous verrons ensemble les éléments fondamentaux d'un programme.

Nom de code : !pac1j00
Clôture du ramassage : 17/11/2019 23:59

Médailles accessibles :

INDEX

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 première 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 règle de compilation particulière.
- Les fichiers seront compilés séparément.

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 :

- putchar

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 – write_alphabet

Fichier à rendre : write_alphabet.c

Pour réussir cet exercice, vous devez afficher l'alphabet en minuscule sur le terminal, suivi d'un saut de ligne. Ci-dessous, vous trouverez les commandes pour compiler votre programme, pour executer votre programme ainsi que la sortie du programme telle qu'attendue.

```
$> ls
write_alphabet.c
$> gcc write_alphabet.c -W -Wall -Werror
$> ./a.out
abcdefghijklmnopqrstuvwxyz
$>
```

Médailles accessibles :

Premier programme !
Vous venez de réaliser le premier programme informatique de votre scolarité.

putchar
Vous avez su utiliser la fonction putchar.

Boucle fixe
Vous êtes parvenu à programmer une boucle simplement bornée.

07 – write_descending_digits

Fichier à rendre : write_descending_digits.c

Cette fois, vous devrez afficher les chiffres de 9 à 0, suivi d'un saut de ligne.

```
$> gcc write_descending_digits.c -W -Wall -Werror
$> ./a.out
9876543210
$>
```


08 – write_halfabet

Fichier à rendre : write_halfabet.c

Nous sommes de retour sur l'alphabet, mais cette fois, nous ne désirons afficher qu'une seule lettre sur deux en partant de la lettre a. Nous finirons encore sur un saut de ligne.

```
$> gcc write_halfabet.c -W -Wall -Werror
$> ./a.out
acegikmoqsuwy
$>
```

Médailles accessibles :

Condition simple
Vous êtes parvenu à écrire une condition simple.

09 – write_waving_alphabet

Fichier à rendre : write_waving_alphabet.c

Dernier exercice sur l'alphabet : cette fois, au lieu de faire disparaître les lettres pair, nous allons les mettre en majuscule. Nous terminerons sur un saut de ligne.

```
$> gcc write_waving_alphabet.c -W -Wall -Werror
$> ./a.out
aBcDeFgHiJkLmNoPqRsTuVwXyZ
$>
```


10 – repeat_char

Fichier à rendre : repeat_char.c

Nous avons écrit assez de programme pour aujourd'hui. Désormais, nous ferons des fonctions à la place. Pour commencer, écrivez la fonction **repeat_char** : cette fonction affiche un caractère passé en paramètre autant de fois qu'indiqué par un second paramètre.

Le prototype de la fonction **repeat_char** est le suivant :

```
void e89_repeat_char(char c, int nbr);
```

Cette fonction doit fonctionner pour toute valeur de c et pour toute valeur positive de nbr. Si nbr est négatif, la fonction ne doit rien afficher.

Le fait que ce programme ne comporte pas de fonction **main** a pour conséquence que vous ne pouvez pas le compiler tout seul. Ce n'est pas un problème pour la correction qui apportera sa propre fonction **main**, mais cela va vous empecher de tester.

Pour pouvoir tester quand même, vous écrirez un « main de test » qui appellera la fonction que vous avez écrit pour pouvoir la tester. Vous ne devrez pas le rendre. Voici par exemple, un main de test pour la fonction repeat_char :

```
int main()
{
	e89_repeat_char('a', 4) ; // affiche aaaa
	e89_repeat_char('z', 3) ; // affiche zzz
	e89_repeat_char('$', 0) ; // n'affiche rien
	e89_repeat_char('_', -1) ; // n'affiche rien
	return (0) ;
}
```

Médailles accessibles :

Procédure
Vous venez d'écrire une fonction ne renvoyant aucune valeur.

Boucle paramétrée
Vous venez d'écrire une boucle dont l'une des bornes n'était pas connue à l'avance :

11 – abs

Fichier à rendre : abs.c

Ecrivez la fonction **e89_abs** qui renvoit la valeur absolue de la valeur reçu en paramètre. Voici quelques exemples de valeur absolue :

- La valeur absolue de 5 est 5.
- La valeur absolue de -8 est 8.

En plus d'écrire la fonction **e89_abs**, vous allez devoir écrire une fonction de test pour la fonction **tc_abs** ! Qu'est ce que ca veut dire ? Que le **TechnoCentre**, le système de correction va fournir une fonction tc_abs dont vous devrez vous assurer du fonctionnement !

Le système de correction s'arrangera pour que parfois, sa fonction marche et parfois pas… votre fonction de test, **test_abs** devra déceler quand est ce qu'elle marche et quand est ce qu'elle ne marche pas ! Elle renverra **vrai** quand la fonction semble fonctionner, et **faux** quand elle detectera qu'elle ne fonctionne pas.

Les prototypes des **e89_abs** et **test_abs** sont les suivants :

```
int e89_abs(int nbr);
bool test_abs();
```

Evidemment, vous pouvez utiliser cette fonction **test_abs** pour tester votre propre fonction **e89_abs** ! L'idée étant ici de vous apprendre à tester !

Pendant que vous travaillez, disposez de votre fonction **main** pour pouvoir tester, ainsi que de fonctions de test testant vos fonctions, et avant de rendre votre travail, effacez votre main et modifiez vos fonctions de test pour qu'elle corrigent celle du TechnoCentre à la place.
Vos fonctions de tests seront alors évalués comme les autres.

Médailles accessibles :

Fonction
Vous venez d'écrire une fonction, avec des paramètres et une valeur de retour.

Test
Vous venez de construire votre premiere fonction de test ! Votre premier pas vers la qualité.

12 – is_printable

Fichier à rendre : is_printable.c

Ecrivez la fonction **e89_is_printable** et la fonction d'évaluation **test_is_printable** qui évaluera **tc_is_printable**. La fonction **e89_is_printable** doit renvoyer vrai si le caractère reçu en paramètre est imprimable. Un caractère est imprimable si il dispose d'un symbole graphique associé, ou si c'est le caractère espace.

Voici le prototype des fonctions :

```
bool e89_is_printable(char c);
bool test_is_printable();
```


13 – is_between

Fichier à rendre : is_between.c

Ecrivez la fonction **e89_is_between** et la fonction d'évaluation **test_is_between** qui évaluera **tc_is_between**. La fonction **e89_is_between** doit renvoyer vrai si le premier de ses paramètres est situé entre son second et son troisième, égalité incluse.

Voici le prototype des fonctions :

```
bool e89_is_between(int a, int b, int c);
bool test_is_between();
```

Attention ! Le paramètre b peut-être plus grand ou plus petit que c…
Votre fonction is_between doit fonctionner dans les deux cas !

14 – clamp

Fichier à rendre : clamp.c

Vous commencez à avoir l'habitude : ecrivez les fonctions pour l'exercice « clamp ». Vous l'aurez deviné, il s'agit de e89_clamp et de test_clamp (testant toujours tc_clamp). Par la suite, nous ne préciserons plus le nom de la fonction de test ni le nom de la fonction testée par la fonction de test : nous supposerons que vous comprendrez par vous-même comment les choses doivent fonctionner.

La fonction clamp doit s'assurer qu'une valeur, passée en première en paramètre est bien située dans un interval donné ensuite. Si cette valeur est plus petite que la valeur minimale, alors la valeur minimale sera renvoyée. Si cette valeur est plus grande que la valeur maximale, alors la valeur maximale sera renvoyée. Si min est plus grand que max, alors la valeur sera toujours renvoyée directement sans traitement.

Voici le prototype des fonctions, dont la fonction de test, précisé pour la dernière fois :

```
int e89_clamp(int a, int min, int max);
bool test_clamp();
```

15 – xor

Fichier à rendre : xor.c

Ecrivez la fonction suivante et les tests associés :

```
bool e89_xor(bool a, bool b);
```

Cette fonction doit effectuer une opération logique « OU EXCLUSIF » entre a et b. Pour que cette opération soit vraie, il faut qu'un seul des deux paramètres soit vrai. Dans tous les autres cas, la fonction renverra faux.

Médailles accessibles :

Logique XOR
Vous avez saisi le sens du OU EXCLUSIF.

16 – rot_minuschar

Fichier à rendre : rot_minuschar.c

Ecrivez la fonction suivante et les tests associés :

```
char e89_rot_minuschar(char c, int n);
```

Cette fonction fait avancer ou reculer le caractère dans c de n lettres. Par exemple :

- Si c vaut 'e' et n vaut 3, le résultat sera 'h'.
- Si c vaut 'e' et n vaut -3, le résultat sera 'b'.
- Si c vaut 'z' et que n vaut 1, le résultat sera 'a'.
- Si a vaut 'a' et que n vaut -1, le résultat sera 'z'.

Si c n'est pas une lettre en minuscule, la fonction renverra immédiatement la valeur contenu dans c.

Médailles accessibles :

Opérateur modulo
Vous avez saisi le sens de l'opérateur modulo.

17 – rclamp

Fichier à rendre : rclamp.c

Ecrivez la fonction suivante et les tests associés :

```
char e89_rclamp(int a, int min, int max);
```

Cette fonction renvoit max si a est plus petit que min. Elle renvoit min si a est plus grand que max et renvoit a dans tous les autres cas.

18 – write_ascii_table

Fichier à rendre : write_ascii_table.c

Ecrivez un programme affichant la table ascii de la façon suivante :

```
$> gcc write_ascii_table.c -W -Wall -Werror
$> ./a.out
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
$>
```

Vous remarquerez que le premier symbole est l'espace ' ' et que le dernier est '~'. Faites attention à l'endroit ou sont situés les sauts de lignes.

19 – mod

Fichier à rendre : mod.c

Ecrivez la fonction **e89_mod**, ainsi que sa fonction de test **test_mod**, testant la fonction **tc_mod** du système de correction. L'objectif de la fonction mod est de renvoyer le reste de la division entre les deux paramètres.

Voici le prototype des fonctions :

```
int e89_mod(int a, int b);
bool test_mod();
```

Entre nombres entiers, les divisions sont dites entières, cela signifie que par exemple, 3 divisé par 2 donne comme résultat 1 et non 1,5. Comme le résultat est 1 et non 1,5, il y a donc un reste après la division. En l'occurence, le reste ici est de 2, car 3 – 3 / 2 vaut… 2.

Il existe un opérateur permettant d'effectuer cette opération. Vous pouvez également vous en passer, bien entendu, mais gardez à l'esprit que cet opérateur vous sera fort utile par la suite.

20 – write_truth

Fichier à rendre : write_truth.c

Ecrivez la fonction **e89_write_truth**. Cette fonction écrit 'y' si son paramètre est vrai, sinon elle écrit 'n'.

```
void e89_write_truth(int n);
```

21 – write_subpart

Fichier à rendre : write_subpart.c

Ecrivez la fonction e89_write_subpart.

```
void e89_write_subpart(char a, char b);
```

Cette fonction écrit tous les caractères situés entre a et b. Si b est plus grand que a, aucun caractère ne doit être écrit.

22 – is_even

Fichier à rendre : is_even.c

Ecrivez la fonction **e89_is_even** ainsi que sa fonction de test.

```
bool e89_is_even(int n);
```

Cette fonction renvoit vrai si n passé en paramètre est pair. Sinon elle renvoit faux.

23 – affine

Fichier à rendre : affine.c

Ecrivez la fonction **e89_affine** ainsi que sa fonction de test associé.

```
int e89_affine(int a, int x, int b);
```

Cette fonction renvoit le résultat de l'opération a * x + b.

24 – distance

Fichier à rendre : distance.c

Ecrivez la fonction **e89_distance** ainsi que sa fonction de test associé.

```
int e89_distance(int a, int b);
```

Cette fonction renvoit la difference absolue entre a et b. Par exemple, entre 0 et 3, il y a 3 de distance. Entre -2 et 2, il y a 4.