

Le but de ce TP est non seulement d'apprendre à compiler et programmer en C, mais aussi de prendre de bonnes habitudes concernant votre organisation dans un système de fichiers sur votre ordinateur. On prendra soin de respecter toutes les consignes concernant la création de fichiers, et de ne pas se précipiter sur la partie "code".

1 Hello World!

1. Ouvrir votre explorateur de fichiers, et créer un nouveau dossier nommé TP08 dans un endroit approprié (pas le Bureau, pas le dossier Téléchargements).

Remarque : si vous êtes sous Windows avec WSL, il faut créer votre dossier à une adresse accessible pour WSL. Si vous êtes dans ce cas, il faut :

- taper la commande suivante dans la barre d'adresse de votre explorateur de fichiers : `\\wsl$`
 - aller dans le dossier : `debian` -> `home` -> `votre_nom_d_utilisateur`
 - si un tel dossier n'existe pas, essayer le dossier : `debian` -> `root`
2. Créer un fichier `hello.c` dans ce dossier TP08.
 3. Écrire dans ce fichier un programme permettant d'écrire la phrase : "Hello World!" (avec un retour à la ligne à la fin).
 4. Ouvrir un terminal, et se déplacer dans le dossier TP08.
 5. Compiler votre programme avec la commande suivante :

```
$ gcc -Wall -pedantic -Werror hello.c -o mon_executable
```

6. Exécuter le fichier obtenu avec la commande suivante :

```
$ ./mon_executable
```

2 Premiers programmes

7. Écrire un programme `carre.c` ayant la structure suivante :
 - une fonction `void afficher_carre(int n)`; qui affiche à l'écran un carré de hauteur n composé d'étoiles (*);
 - une fonction `int main()`; qui demande à l'utilisateur d'entrer une valeur de n , puis exécute `afficher_carre(n)`.
8. Écrire un programme `diagonales.c` ayant la structure suivante :
 - une fonction `void afficher_diagonales(int n, int m)`; qui affiche à l'écran un carré de hauteur n et de largeur m composé d'étoiles (*) et de plus (+) qui alternent pour former des diagonales;
 - une fonction `int main()`; qui demande à l'utilisateur d'entrer des valeurs de n et m , puis exécute `afficher_diagonales(n,m)`.

```
$ gcc carre.c -o exe_carre
$ ./exe_carre
Entrer une valeur de n : 5
*****
*****
*****
*****
*****
```

```
$ gcc diagonales.c -o exe_diag
$ ./exe_diag
Entrer une valeur de n : 5
*****
+****+
*++++*
+****+
*++++*
+****+
```

9. Écrire une programme `noel.c` affichant un arbre de Noël comme ci-dessous, dont la taille dépend d'une valeur saisie par l'utilisateur.

```

$ gcc noel.c -o sapin
$ ./sapin
Entrez la hauteur de l'arbre de Noël: 5
  *
 ***
*****
*****
*****
*****
|

```

3 La suite de Syracuse

La suite de Syracuse est une suite définie par récurrence par $u_0 \in \mathbb{N}^*$, et :

$$\forall n \in \mathbb{N}^*, u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3 \times u_n + 1 & \text{sinon} \end{cases}$$

10. Créer un fichier `syracuse.c` dans votre dossier TP08.
11. Écrire une fonction `int syracuse(int u)`; prenant en argument une valeur u_n et renvoyant u_{n+1} .
12. Écrire une fonction `int temps_de_vol(int u0)`; prenant en argument une valeur u_0 et renvoyant le plus petit n tel que $u_n = 1$.
13. Montrer que $\forall u_0 \in \llbracket 1, 100000 \rrbracket$, la suite de Syracuse finit par devenir stationnaire, égale à 1.

4 C'est plus, c'est moins

On peut tirer un nombre au hasard $x \in \llbracket 0, n - 1 \rrbracket$ avec le programme suivant :

Tirage aléatoire

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    srand(time(NULL)); // permet d'initialiser la seed
    int x = rand() % n; // tire un nombre avec probabilité uniforme dans [0;n[
    printf("%d\n", x);
}

```

14. Écrire un programme `plus_moins.c` qui tire une valeur au hasard $x \in \llbracket 0, 1000 \rrbracket$, puis la fait deviner à l'utilisateur avec le protocole suivant :
 - on demande une valeur y à l'utilisateur ;
 - si $x < y$, on lui affiche "C'est moins !" et on lui redemande une nouvelle valeur ;
 - si $x > y$, on lui affiche "C'est plus !" et on lui redemande une nouvelle valeur ;
 - si $x = y$, c'est gagné !
15. Écrire un programme `plus_moins_inverse.c` qui inverse les rôles du jeu précédent : l'utilisateur choisit (dans sa tête) un nombre à faire deviner, puis l'ordinateur essaye de le deviner.

5 Nombres premiers

16. Télécharger le fichier `fenelon_tp08.zip` sur la page web du cours, et extraire tous les fichiers qu'il contient dans le dossier TP08, puis ouvrir le fichier `premier.c`.

5.1 Bibliothèque sur les nombres premiers

17. Écrire une fonction naïve `bool est_premier(int n)`; prenant en argument un entier n et qui teste si n est premier en testant tous les diviseurs inférieurs à $\frac{n}{2}$.
18. Quelle est la complexité de votre fonction ?
19. Justifier qu'il suffit de tester les diviseurs inférieurs à \sqrt{n} .
En déduire une amélioration de la complexité de votre fonction.
20. Tester votre programme avec les commandes suivantes :

```
$ gcc premier.c test_premier.c -o test_premier -lm
$ ./test_premier
Le nombre 2 est premier.
Le nombre 3 est premier.
Le nombre 4 n'est pas premier.
Le nombre 5 est premier.
Le nombre 169 n'est pas premier.
```

On décide maintenant d'implémenter l'algorithme du crible d'Érathostène ci-dessous, qui permet de calculer efficacement tous les nombres premiers inférieurs à une valeur $N \in \mathbb{N}^*$ prise en argument.

Algorithme 1 : Crible d'Érathostène

Données : un entier $N \in \mathbb{N}^*$

Résultat : un tableau de booléens `premier` de longueur $N + 1$ tel que

$\forall i \in \llbracket 0, N \rrbracket$, `premier[i]` contient Vrai ssi i est premier

`premier` \leftarrow [`Vrai`, ..., `Vrai`];

`premier[0]` \leftarrow `Faux`;

`premier[1]` \leftarrow `Faux`;

pour i allant de 2 à N **faire**

si `premier[i]` **alors**

pour j allant de i^2 à N par pas de i **faire**

`premier[j]` \leftarrow `Faux`;

renvoyer `premier`;

21. Quelle est la complexité du crible d'Érathostène ? On admettra le résultat suivant :

$$\sum_{\substack{p \text{ premier} \\ 2 \leq p \leq N}} \frac{1}{p} \sim \ln(\ln(N))$$

22. Écrire une fonction `bool* erathostene(int N)`; renvoyant un tableau `premier` de booléens (alloué sur le tas) de longueur $N + 1$ tel que $\forall i \in \llbracket 0, N \rrbracket$, `premier[i]` contient `true` si i est premier (et `false` sinon).
23. Écrire une fonction `int* liste_premiers(int N)`; prenant en argument un entier N et renvoyant un tableau `liste` de taille N dont les premières cases contiennent les nombres premiers inférieurs à N (et le reste des cases contiennent 0).
On utilisera la fonction `erathostene` et on fera attention à ne pas avoir de fuite mémoire.

