Fiche TD & TP n°2 : Boucles et fonctions

Partie TD – Pour 2 séances (seulement pour Majeure Info)

Exercice 1 : Exécutez ce programme pas-à-pas à la main. Que s'affiche-t-il dans la console à la fin ?

```c
#include <stdio.h>
int main()
{
    int i;
    int n=0;

    for (i=5; i>=-5; i--)
    {
        n=n+i;
    }

    printf("La valeur finale de n est %d.\n", n);

    return 0;
}
```

Exercice 2 :

(a) Écrivez un programme qui affiche les entiers de 1 à 10, avec une boucle `for`.
(b) Même question, avec une boucle `while`.
(c) Même question, avec une boucle `do...while`.

Exercice 3 :

(a) Traduire en langage C les deux algorithmes suivants, donnés en pseudo-code :

Demander à l'utilisateur un nombre *n*
Si n<0 :
    Afficher « **n est négatif** »

Sinon :
    Pour i allant de 0 à n :
        Afficher la valeur de *3i*
    Fin pour

Demander à l'utilisateur un nombre n
Si n<=0 :
    Afficher « **n est négatif ou nul** »

Sinon :
    Pour i allant de 0 à n, par pas de 3:
        Afficher la valeur de *i*
    Fin pour

(b) Y'aura-t-il des différences à l'exécution entre les deux programmes ? Si oui lesquelles et dans quels cas ?

Exercice 4 : Écrivez un programme qui affiche la table de multiplication d'un entier *n*. Par exemple, pour *n*=3, le programme affichera :
3x1=3, 3x2=6, 3x3=9, 3x4=12, 3x5=15, 3x6=18, 3x7=21, 3x8=24, 3x9=27, 3x10=30

Exercice 5 :

(a) Écrivez un programme qui demande à l'utilisatrice de saisir au clavier un entier strictement positif, et qui continue ainsi à demander des entiers un par un, jusqu'à ce qu'un entier inférieur ou égal à zéro soit entré, auquel cas on affiche « Fin». Vous utiliserez une boucle `while`. Que se passe-t-il si vous oubliez l'initialisation de l'entier ?

(b) Même question, avec un `do...while`. Est-ce qu'il est nécessaire d'initialiser correctement l'entier ?

(c) Reprenez votre programme de la question (a) ou (b) (choisissez le plus pertinent), et modifiez-le pour qu'il affiche à la fin le plus grand entier qui a été rentré par l'utilisatrice. Si aucun nombre strictement positif n'a été entré, vous pourrez afficher une valeur arbitraire.

(d) (*) Même question que la question précédente, en affichant cette fois le plus petit entier (à l'exclusion du dernier entier qui permet d'interrompre le processus).

(e) (*) Même question que la question (c) en affichant cette fois le plus grand entier rentré parmi tous les entiers impairs. Par exemple si l'utilisatrice tape 4, puis 5, puis 10, puis 1, puis 7, puis 12, puis 3, puis -1, le programme doit lui afficher 7 à la fin. Vous penserez bien à gérer le cas où l'utilisatrice ne rentre que des entiers pairs, auquel cas on affichera un message expliquant la situation, au lieu du plus grand entier.

Exercice 6 : Alice et Bob sont partis dans un pays tropical étudier la prolifération de moustiques. Ils ont réussi à identifier la formule d'évolution du nombre de moustiques d'une année sur l'autre : si on appelle N le nombre de moustiques présents à un moment donné, le nombre passe à N/2 + 50 l'année suivante. Pour gagner du temps lors de leur simulation, Alice a écrit un petit programme. Malheureusement, Bob a modifié le programme sans le vouloir. Il a effacé une ligne (marquée par des pointillés) et introduit trois erreurs dans le programme principal.

(a) Corrigez le programme en trouvant les trois erreurs et en complétant la ligne en pointillés.

(b) Exécutez le programme pas-à-pas à la main, en supposant que l'utilisateur tape 50 puis 98. Que s'affiche-t-il dans la console ?

(c) Même question en supposant que l'utilisateur tape 50 puis 100. Que remarquez-vous ?

(d) (*) Quelle solution proposez-vous pour régler le problème observé à la question précédente ?

```c
#include <stdio.h>

int nbAnnees(int nbDebut, int seuil)
{
    int nbMoustiques=nbDebut;
    int annees=0;
     printf("Calcul en cours...\n");
    while (nbMoustiques<seuil)
    {
        nbMoustiques=nbMoustiques/2+50;
        annees++;
        printf("%d\n", nbMoustiques);
    }
    return annees;
}
```

```c
int main()
{

    int initial=0;
    int cible=0;

    printf("Avec combien de moustiques
commencer?");
    scanf("%d", &initial);

    printf("Combien de moustiques visez-
vous?");
    scanf("%d", cible);

    n=……;
    printf("Il vous faudra %d annees.\n", n)

    return 0;
}
```

Exercice 7 :

a) Ecrivez une fonction `sensInverse` qui prend en argument un entier strictement positif *n* et qui affiche tous les entiers de *n* à 0, en revenant à la ligne après chaque entier affiché. Cette fonction ne renvoie rien.
b) Modifier votre fonction `sensInverse` pour qu'elle renvoie la somme des entiers affichés.
c) Ecrivez un programme principal qui demande à l'utilisateur un entier *n* puis qui affiche tous les entiers de *n* à 0, ainsi que la somme des entiers de *n* à 0. Pensez à utiliser la fonction que vous venez d'écrire.

Exercice 7 : Écrivez une fonction `multipleSept` qui prend en argument un entier *n* et qui renvoie le plus petit entier supérieur ou égal à *n* qui est multiple de 7. Écrivez aussi le prototype de cette fonction.

Exercice 8 : Écrivez une fonction `sommeIntervalle` qui prend en argument deux entiers *n* et *m* et qui renvoie la somme des entiers compris entre *n* et *m*. Écrivez aussi le prototype de cette fonction. Que se passe-t-il si n>m ? Est-ce souhaitable ?

Exercice 9 : En utilisant les fonctions `multipleSept` et `sommeIntervalle`, écrivez une fonction `sommeEntreMultipleSept` qui prend en paramètre deux entiers *a* et *b*, et qui renvoie la somme des entiers compris entre *c* et *d*, où *c* est le plus petit entier supérieur ou égal à *a* qui est multiple de 7, et *d* est le plus petit entier supérieur ou égal à *b* qui est multiple de 7.

Exercice 10 :

a) Écrivez le prototype d'une fonction `power` qui prend en paramètre un entier *a* et un entier *b*, et qui renvoie a^b.

b) Écrivez le code de la fonction `power` en utilisant une boucle `for`.

Partie TP – Pour 2 séances

Rappels :

- Munissez-vous de la fiche *"Pour bien démarrer les TP"* distribuée lors du premier TP, notamment pour avoir à portée de main les instructions de compilation, d'exécution et les bonnes pratiques.
- Chaque portion de code (chaque sous-question) doit être **testée au fur et à mesure** sur **au moins 3 exemples** de valeurs différentes **illustrant tous les cas possibles**.
- N'hésitez pas à rajouter des sauts de ligne dans les affichages votre programme (grâce à la ligne `printf("\n");`) pour améliorer la lisibilité de la console après exécution de votre programme.

Exercice 1 :

(a) Récupérer sur Moodle le fichier **diagonale.c**. Ouvrez-le, lisez-le soigneusement puis répondez aux questions suivantes :
- La fonction diagonale prend un seul argument, appelé *n*, de type `int` : *vrai ou faux ?*
- La fonction diagonale n'a pas de valeur de retour : *vrai ou faux ?*
- La fonction diagonale s'exécute avant le lancement du programme principal : *vrai ou faux ?*
- La fonction diagonale utilise exactement 2 variables locales (sans compter les éventuels paramètres) : *vrai ou faux ?*
- Dans le programme principal, ligne 32, on trouve : `m=diagonale(n)`. Cela va provoquer une erreur car l'on cherche à stocker le résultat de l'appel dans une variable appelée `m`, alors qu'elle devrait s'appeler `nbEspaces`, comme à la ligne 21. *Vrai ou faux ?*

- Dans le programme principal, ligne 32, on trouve `m=diagonale(n).` Remarquez que l'argument passé dans cet appel est *n*, tout comme à la ligne de définition de la fonction (ligne 3). Cela est obligatoire : *vrai ou faux ?*
- Dans le programme principal, ligne 32, on trouve `m=diagonale(n).` Remarquez que l'appel à la fonction contient exactement un argument, tout comme à la ligne de définition de la fonction (ligne 3). Cela est obligatoire : *vrai ou faux ?*
- Prédisez le déroulement de ce programme

(b) Compilez et exécutez le programme. Vérifiez que cela est cohérent avec vos prévisions, et dans le cas contraire, analysez pourquoi.

(c) Copiez-collez la fonction `diagonale` pour créer une deuxième fonction `diagonaleDecroissante` et une troisième fonction `diagonaleInversee`, initialement identiques à la fonction `diagonale`. Modifiez-les pour qu'elles affichent le résultat correspondant aux exemples ci-dessous :

```
diagonaleDecroissante(5)
5
 4
  3
   2
    1
     0
```

Valeur de retour : 15
(Il y a zéro espace devant le 5, et cinq espaces devant le 0)

```
diagonaleInversee(5)
     5
    4
   3
  2
 1
0
```

Valeur de retour : 15
(Il y a cinq espaces devant le 5, et zéro espace devant le 0)

Exercice 2 : Récupérer sur Moodle le fichier **geometrieRectangle.c** et compilez-le. Un certain nombre de warnings et d'erreurs doivent apparaître car le programme est incorrect. Modifiez-le pour qu'il compile et exécute normalement. *Indice : nous avons introduit ici 3 fautes, qui nécessitent de modifier le programme à 4 endroits différents au total.*

Conseils :

- *Commencez **toujours** par le premier message qui s'affiche lors de la compilation, même si cela demande de remonter en arrière dans la console. Une fois que vous pensez avoir résolu la première erreur/warning, recompilez pour vérifier s'il a bien disparu. Si oui, continuez avec le nouveau premier message d'erreur, sinon persistez. Il ne sert souvent à rien de régler le deuxième message si le premier n'a pas été résolu correctement.*
- *Le compilateur indique toujours le numéro de la ligne concernée par le message : utilisez-le !*
- *Lisez les messages en entier, traduisez-les dans votre tête, ils apportent de précieuses précisions.*

Exercice 3 : Écrivez un programme qui lit un entier *n*, et qui affiche les entiers de 1 à *n* (inclus), en utilisant une boucle `for`.

Avez-vous bien pensé à tester toutes vos portions de code sur au moins 3 exemples différents ?

Exercice 4 : Testez le programme suivant :

```
#include <stdio.h>
int main(void) {
     int i = 0;
     while (i<=10) {
          printf("i vaut %d\n", i);
     }
     return 0;
}
```

Pour arrêter un programme en cours d'exécution, vous pouvez utiliser `Ctrl-C`. Comment corriger ce programme pour qu'il affiche tous les entiers de 1 à 10 (inclus) ?

Exercice 5 : Écrivez un programme qui lit un entier *n*, et qui affiche les entiers de 1 à *n* (inclus), en utilisant une boucle `while`. Vous noterez que la version avec le `for` (voir exercice 3) est préférable pour deux raisons :

- la version avec le `while` peut provoquer des boucles infinies en cas d'oubli de l'incrémentation (voir exercice 4)
- la syntaxe du `for` permet de ne pas oublier l'initialisation ou l'incrémentation.

Exercice 6 : Écrivez un programme qui lit un entier *n*, et qui affiche les entiers pairs de 0 à *2n* (inclus), en utilisant une boucle `for`.

Exercice 7 :

(a) Écrivez une fonction `somme` qui prend en argument un entier n et qui renvoie la somme des entiers de 1 à *n* (inclus). Vous pourrez la tester en remarquant que cette somme vaut 6 pour *n*=3, et elle vaut 15 pour *n*=5.

(b) Testez votre fonction précédente pour *n*=0, sans la modifier, et vérifiez que la somme est bien 0. Déduisez-en ce qui se passe lorsque l'ordinateur exécute `for (int i=1; i<=0; i++) { … }`.

Exercice 8 :

(a) Écrivez un programme qui demande, par saisies clavier successives, une suite d'entiers terminée par -1, et qui fait la somme de tous ces entiers (à l'exception du -1). Par exemple, si l'utilisatrice saisit 12, 18, 5, -1, le programme affiche 35. Vous utiliserez une boucle `while` : pourquoi ?

(b) Écrivez un programme qui lit un entier *n*, puis une suite de *n* entiers, et qui fait la somme de tous ces entiers (à l'exception de *n*). Par exemple, si l'utilisatrice saisit 3, 12, 18, 5, le programme affiche 35. Vous utiliserez une boucle `for` : pourquoi ?

(c) Modifiez vos deux programmes précédents pour qu'ils affichent également le plus grand et le plus petit entier qui ont été donnés (on exclura bien sûr le -1 pour la partie (a), ainsi que le premier entier *n* pour la partie (b)).

Exercice 9 :

(a) Écrivez un programme qui lit un entier *n*, et qui affiche une ligne de *n* caractères 'x'. Par exemple, si *n*=5, le programme affiche :

```
xxxxx
```

(b) Écrivez un programme qui lit un entier *n*, et qui affiche un carré de *n* x *n* caractères 'x'. Par exemple, si *n*=3, le programme affiche :

```
xxx
xxx
xxx
```

c) Écrivez un programme qui lit un entier *n*, et qui affiche un triangle rectangle (inscrit dans le carré de *n* x *n* caractères) dont l'angle droit est en bas à gauche. Par exemple, si *n*=3, le programme affiche :

```
x
xx
xxx
```

d) Même question, mais avec un angle droit en haut à gauche :

```
xxx
xx
x
```

e) Même question, mais avec un angle droit en haut à droite :

```
xxx
 xx
  x
```

f) Même question, mais avec un angle droit en bas à droite :

```
  x
 xx
xxx
```

Avez-vous bien pensé à tester toutes vos portions de code sur au moins 3 exemples différents ?

Exercice 10 :

(a) Écrivez une fonction `rectangleFixe` qui prend en argument deux entiers strictement positifs *n* et *m* et qui affiche *n* fois la ligne obtenue par concaténation de tous les entiers de 1 à *m*, séparés par un espace. Par exemple pour *n=3* et *m=5*, on doit obtenir l'affichage suivant :

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

(b) Dans le même esprit, écrivez une fonction `rectangleSomme` qui prend également en argument deux entiers strictement positifs *n* et *m*, et qui affiche la valeur de *i+j* lorsque l'on est sur la $i^{\text{ième}}$ ligne et la $j^{\text{ième}}$ colonne (on commencer à numéroter à partir de 1). Par exemple pour *n=3* et *m=5*, on doit obtenir l'affichage suivant :

```
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
```

(c) Dans le même esprit, écrivez une fonction `rectangleSommeDetail` qui affiche un résultat similaire à la fonction `rectangleSomme` en remplaçant le résultat de *i+j* par l'écriture explicite de l'opération, comme dans l'exemple suivant pour *n=3* et *m=5* :

```
1+1 1+2 1+3 1+4 1+5
2+1 2+2 2+3 2+4 2+5
3+1 3+2 3+3 3+4 3+5
```

Exercice 11 :

(a) Dans un nombre écrit en base 10, on peut extraire le chiffre des unités avec un modulo 10, et on peut retirer le chiffre des unités en faisant une division entière du nombre par 10. Écrivez une fonction `ChiffreParChiffre` qui prend en argument un entier *n*, et qui affiche la liste des chiffres de *n*, en partant des unités. Par exemple, si *n*=3456, le programme affiche 6, puis 5, puis 4, puis 3. Vous utiliserez une boucle `while`. Cette fonction ne renvoie rien.

(b) Écrivez une fonction `nbChiffres` qui prend en argument un entier *n* et qui renvoie le nombre de chiffres dans l'écriture de *n* en base 10.

(c) (*) Testez votre fonction `rectangleSomme` de l'exercice 10 avec des entiers *n* et *m* supérieurs à 10 et observez le décalage qui apparaît à cause de la différence de longueur des nombres. Nous allons tenter de corriger cela.

- Ecrivez une fonction `affiche` qui prend en argument un entier *n* et un entier *longueur*, et qui affiche le nombre *n* en rajoutant des 0 devant, de telle sorte que le nombre total de caractères affichés soit égal à *longueur*. Par exemple, `affiche(5,3)` doit afficher `005`.
- Ecrivez votre fonction `rectangleSommeAligne` pour qu'il n'y ait plus de décalage, même lorsque *n* et *m* sont grands. Vous utiliserez la fonction `affiche` en choisissant soigneusement l'entier *longueur*.

(d) (*) Écrivez une fonction `ChiffreParChiffreOrdonne` qui prend en argument un entier *n*, et qui affiche les chiffres de *n*, dans l'ordre, séparés par un espace. Par exemple, si *n*=3456, le programme affiche 3, puis 4, puis 5, puis 6. L'algorithme à utiliser est le suivant :

$c \leftarrow$ nombre de chiffres de n
$p \leftarrow 10^{c-1}$
tant que $n \neq 0$ faire
    afficher $\lfloor n/p \rfloor$
    $n \leftarrow n - \lfloor n/p \rfloor p$
    $p \leftarrow p/10$
    $c \leftarrow c-1$
fin tant que

Avez-vous bien pensé à tester toutes vos portions de code sur au moins 3 exemples différents ?

Exercice 12 : Soit *n* un entier. L'entier suivant *n* selon une suite de Syracuse est défini de la manière suivante : *suivant(n)*=1 si *n*=1, *suivant(n)*=*n*/2 si *n* est pair, *suivant(n)*=3*n*+1 sinon. Écrivez une fonction `suivant` qui prend en argument un entier *n*, et qui renvoie l'entier suivant dans l'ordre de Syracuse ; puis une fonction Syracuse qui prend en argument un entier n et qui affiche les valeurs de la suite de Syracuse commençant par *n* et en s'arrêtant au premier 1 rencontré. La fonction `Syracuse` ne renvoie rien. Par exemple, si *n*=10, la fonction affiche 10, 5, 16, 8, 4, 2 et 1.

Exercice 13 : Écrivez une fonction `zigzag` qui prend trois arguments : un entier *debut*, un entier *fin* et un entier *nbDeZ*. Cette fonction devra afficher un « zigzag » conformément aux exemples ci-dessous (les nombres pairs doivent écrits à gauche du zigzag, les nombres impairs à droite ; les entiers de début, de fin, et de nombre de z dans l'affichage du zigzag sont réglés par les paramètres du même nom). La fonction doit renvoyer le nombre de lignes écrites. Vous chercherez à optimiser la longueur de votre code pour éviter les répétitions inutiles.

`zigzag(3,10,2)`	`zigzag(-2,4,5)`	`zigzag(13,15,1)`	`zigzag(5,2,3)`

`zzigzag 3`
`4 zzigzag`
`zzigzag 5`
`6 zzigzag`
`zzigzag 7`
`8 zzigzag`
`zzigzag 9`
`10 zzigzag`
	`-2 zzzzzigzag`
	`zzzzzigzag -1`
	`0 zzzzzigzag`
	`zzzzzigzag 1`
	`2 zzzzzigzag`
	`zzzzzigzag 3`
	`4 zzzzzigzag`
		`zigzag 13`
		`14 zigzag`
		`zigzag 15`	

Renvoie : `8`	Renvoie : `7`	Renvoie : `3`	Renvoie : `0`

Avez-vous bien pensé à tester toutes vos portions de code sur au moins 3 exemples différents ?

Si vous avez fini le TP en avance : reprenez les exercices du TD et faites-les sur machine.