



## Exercices de programmation en C- ue AAGB

### Partie 1: syntaxe de base

Exercices provenant en partie du module C avancé LU2IN018

Version du 21 septembre 2022

Exercices à faire :

- Premier programme
- Types de base, affichage, scanf et division entière
- Comparaison de flottants
- Le type **char** est un entier comme les autres
- Première fonction : Convertisseur francs-euros
- Fin séance 1
- Syntaxe des alternatives : qui est le plus grand ?
- Les 3 boucles : Factorielles
- Boucles imbriquées : Nombres premiers
- Boucle **while** avec condition complexe : Racine Carrée
- Fin séance 2

## Table des matières

<b>1 Premiers pas</b>	<b>2</b>
b Premier programme . . . . .	2
b Types de base, affichage, scanf et division entière . . . . .	2
b Comparaison de flottants . . . . .	3
b Le type <b>char</b> est un entier comme les autres . . . . .	3
<b>2 Les fonctions</b>	<b>4</b>
b Première fonction : Convertisseur francs-euros . . . . .	4
<b>3 Alternatives, Boucles</b>	<b>4</b>
b Syntaxe des alternatives : qui est le plus grand ? . . . . .	4
b Les 3 boucles : Factorielles . . . . .	4
b Boucles imbriquées . . . . .	4
b Boucle <b>while</b> avec condition complexe : Racine Carrée . . . . .	5
<b>4 Exercices supplémentaires</b>	<b>5</b>
a Syntaxe des alternatives et <b>switch</b> : base complémentaire . . . . .	5
e Boucles imbriquées : Nombres premiers . . . . .	6
e Boucle <b>while</b> et traduire un algorithme : Calcul de PGCD . . . . .	6
e Calcul de pi . . . . .	7

## Niveau des exercices

Les exercices sont répartis selon trois niveaux :

- base : ces exercices couvrent les notions de base et sont à faire obligatoirement.
- entraînement : ces exercices abordent des notions déjà présentées dans d'autres exercices. Il est recommandé de les faire pour s'exercer.
- approfondissement : ces exercices sont plus difficiles. Il est intéressant de les faire pour aller plus loin.

## 1 Premiers pas

### Exercice 1 (*base*) – Premier programme

#### Notions abordées

- ★ Ecrire un programme en C
- ★ Compilation
- ★ Les bibliothèques (`#include`)
- ★ trouver quelle(s) bibliothèque(s) inclure

Pour commencer, ouvrez un Terminal et lancez un éditeur de texte (c'est-à-dire tapez `gedit`, `emacs` ou `vim`, sans oublier le `&` à la fin). Vous écrirez vos programmes C dans cet éditeur, en les sauvegardant sous un nom explicite se terminant toujours par l'extension `.c` (ou `.h`).

1. Recopiez le programme suivant, compilez le (il faudra taper la commande `gcc . . .` dans le terminal) et exécutez le (il faudra taper `./MonExecutable` dans le terminal).

```
1
2 int main() {
3     int annee=2019;
4
5     return 0;
6 }
```

Il faut ensuite compiler votre programme. Pour compiler vos programme, vous utiliserez le compilateur `gcc` comme suit :

```
gcc -Wall -o MonExecutable monProgramme.c
```

Le programme exécutable nommé `MonExecutable` est créé à partir de votre programme que vous avez écrit dans le fichier texte `monProgramme.c`. Vous pourrez l'exécuter en tapant dans le terminal `./MonExecutable`

2. Ajoutez la ligne `printf("Hello world in %d\n", annee);` puis compilez à nouveau.

Le programme ne compile pas et l'erreur "warning: implicit declaration of function 'printf'" est affichée. Cela signifie que le compilateur ne trouve pas la fonction `printf` car la bibliothèque la définissant n'est pas incluse dans votre programme.

3. Tapez `man 3 printf` dans le terminal. Déterminez quelle bibliothèque est nécessaire puis ajoutez là à votre code. Compilez et exécutez à nouveau votre programme.

**Exercice 2 (base) – Types de base, affichage, scanf et division entière****Notions abordées**

- ★ les types
- ★ la fonction `printf`
- ★ la fonction `scanf`
- ★ la division entière

La fonction `scanf` permet de récupérer ce qui est tapé au clavier par l'utilisateur. Son prototype est :

**int** `scanf(const char * restrict format, ...)` ;

Le "format" est le même qu'avec la fonction `printf`, et il faut ensuite donner l'**adresse** des variables dans lesquelles il faut mettre les valeurs lues (nous y reviendront plus tard). Par exemple, pour lire un entier, la fonction `scanf` s'écrit :

```
1 int i1 ;  
2 scanf ("%d" , &i1 ) ;
```

L'opérateur `&` permet d'avoir l'adresse de la variable `i1` ; il ne faut surtout pas l'oublier. Il est conseillé d'écrire des formats les plus simples possibles car sinon le comportement de la fonction `scanf` devient rapidement difficile à comprendre.

1. Ecrire un programme qui demande un entier positif à l'utilisateur et l'affiche au format entier (digit `%d`), octal (`%o` et hexadécimal (`%X`).
2. Compléter le programme précédent pour qu'il demande un autre entier à l'utilisateur (différent de 0) et affiche le résultats de la division du premier par le second. Que remarquez vous ? Comment l'éviter ?

**Exercice 3 (base) – Comparaison de flottants****Notions abordées**

- ★ comparaison de **float**

1. Savez vous pourquoi on ne peut tester l'égalité entre nombres de type **float** ? Nous allons l'illustrer avec un exemple. Compléter le programme précédent en déclarant 2 variables de type **float** initialisées à 0.1. Ecrire ensuite un **if** qui teste si la somme de ces 2 variables est égale à 0.2 ou non. Que remarquez vous ? Faites le même test avec des variables initialisées à 2 (et la somme à 4).
2. Modifiez le programme précédent pour vérifier non plus que la somme des deux variables initialisées à 0.1 est égale à 0.2 mais plutôt que la différence est inférieure à  $10^{-6}$  (par exemple). Le comportement e votre programme vous semble-t-il plus cohérent ? Ceci est la bonne manière de comparer des **float**.

**Exercice 4 (base) – Le type char est un entier comme les autres****Notions abordées**

- ★ le type **char**
- ★ le code ASCII

1. Ecrire un programme qui demande à l'utilisateur un caractère et affiche son code ASCII, ainsi que la caractère suivant et son code ASCII et le caractère précédent et son code ASCII. Dans ce programme, vous déclarerez aussi une autre variable de type `char` initialisée à 127 et vous l'afficherez comme un entier (`%d`) avant et après l'avoir incrémentée de 1. Que remarquez vous ?

## 2 Les fonctions

### Exercice 5 (*base*) – Première fonction : Convertisseur francs-euros

1. Écrivez une fonction convertir qui prend en argument un nombre flottant et un caractère (e ou f) et qui renvoie la valeur du nombre convertie de francs en euros (resp. d'euros en francs) lorsque le caractère est f (resp. e).  
Vous écrirez une fonction main pour la tester

## 3 Alternatives, Boucles

### Exercice 6 (*base*) – Syntaxe des alternatives : qui est le plus grand ?

#### Notions abordées

- ★ if et else
- ★ la fonction scanf

Prenons un programme manipulant deux variables de type `int` a et b :

```
#include <stdio.h>

int main() {
    int a, b;

    printf("Entrez une valeur pour a\n");
    scanf("%d", &a);
    printf("Entrez une valeur pour b\n");
    scanf("%d", &b);

    return 0;
}
```

1. Complétez ce programme pour qu'il affiche "a est plus grand que b" si a est supérieur à b.
2. Complétez le programme précédent pour qu'il affiche "b est au moins aussi grand que a" si a n'est pas supérieur à b.

### Exercice 7 (*base*) – Les 3 boucles : Factorielles

La factorielle du nombre  $n$ , notée  $n!$ , est définie par  $n! = 1 \times 2 \times \dots \times (n - 1) \times n$ . Par convention,  $0! = 1$ .

1. Écrivez un programme qui calcule la factorielle du nombre 10 en utilisant une boucle for.
2. Écrivez un programme qui calcule la factorielle du nombre 10 en utilisant une boucle while.
3. Écrivez un programme qui calcule la factorielle du nombre 10 en utilisant une boucle do-while.
4. Y a-t-il une différence à l'exécution entre les boucles for/while et la boucle do-while ?
5. Dans cet exemple de calcul de la factorielle, quel type de boucle vous semble le plus approprié ?

**Exercice 8 (base) – Boucles imbriquées**

1. Ecrivez un programme qui affiche toutes les tables de multiplications de 1 à 10 de manière lisible.

**Exercice 9 (base) – Boucle while avec condition complexe : Racine Carrée**

L'algorithme de Héron d'Alexandrie (ou de Newton) propose une méthode itérative pour calculer la racine carrée d'un nombre strictement positif  $a$ . Soit  $x$  la racine carrée de ce nombre  $a$ . On a :

$$\begin{aligned}x^2 &= a \\x^2 + x^2 &= x^2 + a \\2x &= (x^2/a)/x \\x &= (x^2 + a)/2x \\x_{n+1} &= \frac{x_n + \frac{A}{x_n}}{2}\end{aligned}$$

D'après le théorème du point fixe<sup>1</sup>, on obtient alors la suite :  $x_{i+1} = (x_i + a/x_i)/2$ .

Le calcul peut être commencé avec n'importe quelle valeur  $x_0$  différente de zéro. Par exemple, pour la racine de 2 ( $a = 2$ ) en partant de  $x_0 = 1$  :

$$\begin{aligned}x_0 &= 1 \\x_1 &= (1/2)(1 + 2/1) = 3/2 = 1,5 \\x_2 &= (1/2)(3/2 + 2/(3/2)) = 17/12 = 1,4166666 \\x_3 &= (1/2)(17/12 + 2/(17/12)) = 577/408 = 1,41421568\end{aligned}$$

1. Écrivez un programme qui calcule la racine carrée d'une valeur  $a$  demandée à l'utilisateur en appliquant cet algorithme en calculant jusqu'à  $x_{10}$ .
2. On souhaite arrêter le calcul lorsque la différence relative entre deux valeurs calculées (c'est-à-dire  $(x_{i+1} - x_i)/x_i$ ) est inférieure en valeur absolue à une précision  $p$  que l'on s'est fixée (NB :  $|x| < p$  équivaut à  $p < x < p$ ). Modifier votre programme pour que le nombre d'itérations correspondent à cette condition.

**4 Exercices supplémentaires****Exercice 10 (approfondissement) – Syntaxe des alternatives et switch : base complémentaire Notions abordées**

★ switch

★ scanf

---

1. Soit  $h$  une fonction dérivable, de fonction dérivée  $h'$  continue sur un voisinage  $V$  d'un point fixe  $b$  de  $h$ , c'est-à-dire tel que  $h(b) = b$  et vérifiant :  $\sup_{x \in V} |h'(x)| < 1$ , alors la suite  $(x_n)$  définie par  $x_{n+1} = h(x_n)$  converge vers  $b$  dès qu'il existe  $N$  tel que  $x_N$  est dans  $V$ . Ici,  $h(x) = (x + a/x)/2$  et le point fixe est  $b = \sqrt{a}$ .

Soit un programme demandant une lettre à l'utilisateur.

```
#include <stdio.h>

int main() {
    char a;

    printf("Donnez un nucléotide: \n");
    scanf("%c", &a);

    return 0;
}
```

1. Modifiez ce programme pour qu'il affiche la base complémentaire de celle entrée par l'utilisateur et une message d'erreur ("base invalide") si ce n'est pas une base nucléotidique. Vous utiliserez des alternatives (`if`).
2. Ecrivez le même programme cette fois en utilisant un `switch`

### Exercice 11 (*entraînement*) – Boucles imbriquées : Nombres premiers

Nous souhaitons afficher les nombres premiers inférieurs à  $N$  et supérieurs à 1.

Un nombre premier est un nombre exclusivement divisible par 1 et lui-même. Pour décider si un nombre  $M$  est premier, nous allons utiliser une variable entière `premier` initialisée à 1. Nous allons parcourir tous les nombres entre 2 et  $M$ . Si l'on trouve parmi ces nombres un diviseur de  $M$ , alors la variable `premier` est mise à 0. Si la variable `premier` a toujours la valeur 1 à la fin du parcours, alors  $M$  est premier.

Cet ensemble d'opérations doit être exécuté pour tous les nombres  $M$  entre 2 et  $N$ .

1. Pour commencer, écrivez un programme C qui permet de déterminer si  $M$ , nombre demandé à l'utilisateur, est premier ou non, en recherchant un diviseur entre 2 et  $M$ . Votre programme affichera à l'écran "M est premier" ou "M n'est pas premier" selon la valeur de `premier` à l'issue de votre boucle.
2. Traduisez maintenant l'algorithme complet en un nouveau programme C qui affiche tous les nombres premiers compris entre 2 et  $N$  ( $N$  est demandé à l'utilisateur).
3. Dans notre programme, la boucle `for` parcourt tout l'intervalle  $[2, N[$  à la recherche d'un diviseur, alors qu'il serait plus judicieux de s'arrêter dès qu'on a trouvé un diviseur. Et même lorsque  $N$  est premier, il n'est pas nécessaire de vérifier sur tout l'intervalle : si on n'a pas trouvé de diviseur plus petit que  $N$ , alors  $N$  est premier.

Modifiez votre programme afin de prendre en compte ces remarques. et comparez les temps d'exécution avec ceux obtenus à la question précédente, et vérifiez que ce nouveau programme est plus efficace.

### Exercice 12 (*entraînement*) – Boucle while et traduire un algorithme : Calcul de PGCD

On souhaite déterminer le PGCD de deux nombres en appliquant l'algorithme suivant :

```
tant que (nombre_1 different de nombre_2)
faire
    si nombre_1 > nombre_2
    alors
        nombre_1=nombre_1-nombre_2
```

```
    sinon
nombre_2=nombre_2-nombre_1
    fin si
fin tant que
```

1. Écrivez un programme C calculant le PGCD de deux nombres  $N1$  et  $N2$ .

### Exercice 13 (*entraînement*) – Calcul de $\pi$

1. Le nombre  $\pi$  peut être calculé en appliquant la suite de Leibniz :  $\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - \dots$   
Supposons que l'on veuille calculer  $\pi$  avec une précision  $p$  (*i.e.* nous souhaitons calculer une approximation  $pi$  de  $\pi$  telle que  $\pi - p \leq pi \leq \pi + p$ ). Quel type de boucle vous semble la plus appropriée pour calculer  $\pi$  ?
2. Quel algorithme proposez-vous ? Ecrivez-le en C.