
| Extrait du référentiel : BTS Systèmes Numériques option A (Informatique et Réseaux) | | Niveau(x) |
|---|---|---|
| S4. Développement logiciel<br>S4.4. Programmation procédurale | Manipulations de données (« quoi ») en pseudo-langage et/ou en langage C<br>Transcription d'algorithmes (« comment ») en pseudo-langage et/ou en langage C<br>Développement de programmes « console » avec gestion des arguments de la ligne de commande | 4<br>4<br>3 |

## Objectifs du cours :

- La boucle while
- La boucle do…while
- La boucle for :
  - plusieurs compteurs
  - imbrication
- Exercices

Une **boucle** est un moyen de répéter des instructions suivant le résultat d'une condition. Ces structures, dîtes **itératives**, que nous allons sont les suivantes :

**while** : répète une suite d'instructions tant qu'une condition est respectée.
**do**…**while** : répète une suite d'instructions tant qu'une condition est respectée. Le groupe d'instructions est exécuté au moins une fois.
**for** : répète un nombre fixé de fois une suite d'instructions.

# LA BOUCLE WHILE

La syntaxe est la suivante :

```
while (/* Condition */)
{
      /* Bloc d'instructions à répéter */
}
```

Si la condition n'est pas vérifiée, le bloc d'instructions est passé et le programme recommence immédiatement à la suite du bloc d'instructions.

Exemple :

```
#include <stdio.h>

int main(void)
  {
      int i = 0;

      while (i < 5)
      {
          printf("La variable i vaut %d\n", i);
          i++;
      }

      return 0;
}
```

Résultat :

```
La variable i vaut 0
La variable i vaut 1
La variable i vaut 2
La variable i vaut 3
La variable i vaut 4
```

Le fonctionnement est simple :

Au départ, notre variable **i** vaut zéro. Étant donné que zéro est bien inférieur à cinq, la condition est vraie, le corps de la boucle est donc exécuté.
La valeur de **i** est affichée.
**i** est augmentée d'une unité et vaut désormais un.
La condition de la boucle est de nouveau vérifiée.
Ces étapes vont ainsi se répéter pour les valeurs un, deux, trois et quatre. Quand la variable **i** vaudra cinq, la condition sera fausse, et l'instruction while sera alors passée.

Dans cet exemple, nous utilisons une variable nommée **i**. Ce nom lui vient d'une contraction du mot anglais **iterator** qui signifie que cette variable sert à l'itération (la répétition) du corps de la boucle.

## EXERCICE

***Question***

***Réalisez un programme qui détermine si un nombre entré par l'utilisateur est premier. Pour rappel, un nombre est dit premier s'il n'est divisible que par un et par lui-même. Notez que si un nombre xx est divisible par yy alors le résultat de l'opération x % y est nul.***

Pour savoir si un nombre est premier, il va vous falloir vérifier si celui-ci est uniquement divisible par un et lui-même. Autrement dit, vous allez devoir contrôler qu'aucun nombre compris entre 1 et le nombre entré (tout deux exclus) n'est un diviseur de ce dernier. Pour parcourir ces différentes possibilités, une boucle va vous être nécessaire.

## LA BOUCLE DO…WHILE

À la différence de la boucle while, la condition est placée à la fin du bloc d'instruction à répéter, ce qui explique pourquoi celui-ci est toujours exécuté au moins une fois. Remarquez également la présence d'un point-virgule à la fin de l'instructions qui est obligatoire.

La syntaxe est la suivante :

```
do
{
    /* Bloc d'instructions à répéter */
} while (/* Condition */);
```

Exemple 1 :

```
#include <stdio.h>

int main(void)
{
    int i = 0;

    do
    {
        printf("La variable i vaut %d\n", i);
        ++i;
    } while (i < 5);

    return 0;
}
```

Résultat :

```
La variable i vaut 0
La variable i vaut 1
La variable i vaut 2
La variable i vaut 3
La variable i vaut 4
```

Exemple 2 :

```
#include <stdio.h>

int main(void)
{
    do
        printf("Boucle do-while\n");
    while (0);

    return 0;
}
```

Résultat :

```
Boucle do-while
```

Comme nous pouvons le voir, bien que la condition soit fausse (pour rappel, une valeur nulle correspond à une valeur fausse), le corps de la boucle est exécuté une fois puisque la condition n'est évaluée qu'après le parcours du bloc d'instructions.

## LA BOUCLE FOR

La syntaxe est la suivante :

```
for (/* Expression/Déclaration */; /* Condition */; /* Expression */)
{
    /* Instructions à répéter */
}
```

Une boucle **for** se décompose en trois parties :

- une expression et/ou une déclaration qui sera le plus souvent l'initialisation d'une variable ;
- une condition ;
- une seconde expression, qui consistera le plus souvent en l'incrémentation d'une variable.

Exemple :

```
#include <stdio.h>

int main(void)
{
    for (int i = 0; i < 5; ++i)
        printf("La variable i vaut %d\n", i);

    return 0;
}
```

Résultat :

```
La variable i vaut 0
La variable i vaut 1
La variable i vaut 2
La variable i vaut 3
La variable i vaut 4
```

La déclaration et l'initialisation de la variable **i** est située en dehors du corps de la boucle. Elle n'est donc pas exécutée à chaque tour.

## EXERCICE

***Question***

***Réalisez un programme qui calcule la somme de tous les nombres compris entre un et nn (nn étant déterminé par vos soins). Autrement dit, pour un nombre nn donné, vous allez devoir calculer 1 + 2 + 3 + ... + (n-2) + (n-1) + n1+2+3+...+(n−2)+(n−1)+n.***

Plusieurs compteurs :

Notez que le nombre de compteurs, de déclarations ou de conditions n'est pas limité.

```
for (int i = 0, j = 2; i < 10 && j < 12; i++, j += 2)
```

Ici, nous définissons deux compteurs **i** et **j** initialisés respectivement à zéro et deux. Le contenu de la boucle est exécuté tant que **i** est inférieur à dix et que **j** est inférieur à douze, **i** étant augmentée de une unité et **j** de deux unités à chaque tour de boucle. Le code est encore assez lisible, cependant la modération est de mise, un trop grand nombre de paramètres rendant la boucle **for** illisible.

Imbrications :

Il est possible d'imbriquer une ou plusieurs boucles en plaçant une boucle dans le corps d'une autre boucle.

Exemple : une liste de nombres dont le produit vaut mille

```
#include <stdio.h>

int main(void)
{
    for (int i = 0; i <= 1000; ++i)
        for (int j = i; j <= 1000; ++j)
            if (i * j == 1000)
                printf ("%d * %d = 1000 \n", i, j);

    return 0;
}
```

Résultat :

```
1 * 1000 = 1000
```

```
2 * 500 = 1000
4 * 250 = 1000
5 * 200 = 1000
8 * 125 = 1000
10 * 100 = 1000
20 * 50 = 1000
25 * 40 = 1000
```

Vous n'êtes bien entendu pas tenu d'imbriquer des types de boucles identiques. Vous pouvez parfaitement plaçer, par exemple, une boucle **`while`** dans une boucle **`for`**.

# EXERCICES

## EXERCICE N°1

Au treizième siècle, un mathématicien italien du nom de Leonardo Fibonacci posa un petit problème dans un de ses livres, qui mettait en scène des lapins. Ce petit problème mit en avant une suite de nombres particulière, nommée la suite de Fibonnaci, du nom de son inventeur.
Il fit les hypothèses suivantes :

- le premier mois, nous plaçons un couple de lapins dans un enclos ;
- un couple de lapin ne peut procréer qu'à partir du troisième mois de sa venue dans l'enclos (autrement dit, il ne se passe rien pendant les deux premiers mois) ;
- chaque couple capable de procréer donne naissance à un nouveau couple ;
- enfin, pour éviter tout problème avec la SPA, les lapins ne meurent jamais.

***Question***

***Le problème est le suivant : combien y a-t-il de couples de lapins dans l'enclos au n-ième mois ? Le but de cet exercice est de réaliser un petit programme qui fasse ce calcul automatiquement.***

## EXERCICE N°2

Connaissez-vous le roman « La Disparition » ? Il s'agit d'un roman français de Georges Perec, publié en 1969. Sa particularité est qu'il ne contient pas une seule fois la lettre « e ». On appelle ce genre de textes privés d'une lettre des lipogrammes. Celui-ci est une prouesse littéraire, car la lettre « e » est la plus fréquente de la langue française : elle représente une lettre sur six en moyenne ! Le roman faisant environ trois cents pages, il a sûrement fallu déployer des trésors d'inventivité pour éviter tous les mots contenant un « e ».
Si vous essayez de composer un tel texte, vous allez vite vous rendre compte que vous glissez souvent des « e » dans vos phrases sans même vous en apercevoir.

Nous avons besoin d'un vérificateur qui nous sermonnera chaque fois que nous écrirons un « e ». C'est là que le langage C entre en scène !

***Question***

***Écrivez un programme qui demande à l'utilisateur de taper une phrase, puis qui affiche le nombre de « e » qu'il y a dans celle-ci. Une phrase se termine toujours par un point « . », un point d'exclamation « ! » ou un point d'interrogation « ? ». Pour effectuer cet exercice, il sera indispensable de lire la phrase caractère par caractère.***