# Chapitre 5

# Structure de boucle : while / do...while

Dans cette seconde partie consacrée aux structures itératives, nous aborderons les deux autres formes qu'offre le langage C : les boucles pour lesquelles le nombre d'itérations est inconnu à l'avance (à l'entrée dans la boucle).

## 1 La boucle FAIRE ... TANT QUE : "do ... while"

Cette forme permet de construire une structure répétitive dans laquelle la condition de rebouclage est vérifiée à la fin : **on est donc certain d'exécuter au moins une fois le bloc d'instruction à répéter.**

Syntaxe :

```
do {
    bloc d'instructions à répéter
} while (condition de rebouclage) ;
```

**N.B.** : attention au point-virgule (";") après la clause "while".

Exemple :

```
/* Programme pour tester la structure "do...while" :
   - boucle 10 fois en affichant une valeur i incrementee a chaque iteration
   - affiche la valeur de i apres la derniere boucle.
*/
#include <stdio.h>

int main () {
    int i = 0 ;
    do {
        printf ("iteration %d \n", i) ;
        i = i + 1 ;
    } while ( i < 10 ) ;
    printf ("valeur de i apres la boucle : %d \n", i) ;

    return 0 ;
}
```

## 2 La boucle TANT QUE : "while"

Cette deuxième forme est très similaire à la précédente exceptée qu'elle permet de construire une structure pour laquelle le bloc d'instructions à répéter peut éventuellement n'être jamais exécuté (comme dans le cas de la structure itérative "for") car la condition est vérifiée avant le bloc.

Syntaxe :

```
while (condition de boucle) {
    bloc d'instructions à répéter
}
```

Exemple :

```
/* Programme pour tester la structure "while" :
   - boucle 10 fois en affichant une valeur i incrementee a chaque iteration
   - affiche la valeur de i apres la derniere boucle.
*/
#include <stdio.h>

int main () {
    int i = 0 ;
    while ( i < 10) {
        printf ("iteration %d \n", i) ;
        i = i + 1 ;
    }
    printf ("valeur de i apres la boucle : %d \n", i) ;
    return 0 ;
}
```

# 3 Exercices

Rappels : on peut combiner plusieurs conditions à l'aide des opérateur "&&" (ET) et || (OU).

### Question 5-1 Vérification des notions de base → exercice de cours

1. Reprendre les deux exemples (1 et 2) du cours et vérifier le bon déroulement des deux programmes. Ecrire et compiler :

```
/* Programme pour tester la structure "while" :
   - boucle 10 fois en affichant une valeur i incrementee a chaque iteration
   - affiche la valeur de i apres la derniere boucle.
*/
#include <stdio.h>

int main () {
    int i = 0 ;
    while ( i < 10) {
        printf ("iteration %d \n", i) ;
        i = i + 1 ;
    }
    printf ("valeur de i apres la boucle : %d \n", i) ;
    return 0 ;
}
```

Ecrire et compiler :

```
/* Programme pour tester la structure "do...while" :
   - boucle 10 fois en affichant une valeur i incrementee a chaque iteration
   - affiche la valeur de i apres la derniere boucle.
*/
#include <stdio.h>

int main () {
    int i = 0 ;
    do {
        printf ("iteration %d \n", i) ;
        i = i + 1 ;
    } while ( i < 10 ) ;
    printf ("valeur de i apres la boucle : %d \n", i) ;

    return 0 ;
}
```

2. Que se passe-t-il si vous "oubliez" l'instruction $\mathbf{i = i + 1}$ ?
3. Initialisez $i$ à 10 dans les deux programmes, que se passe-t-il ? Une boucle infinie se produit...
4. Pourquoi ? La variable $i$ ne s'incrémentant jamais, la condition $i < 10$ sera **toujours** respectée. Le programme ne s'arrêtera que lorsque l'utilisateur (ou l'administrateur) le détruira en lui envoyant un signal de terminaison par exemple.

**Note :** Forcer l'arrêt d'un programme en cours d'exécution se fait à l'aide des touches Ctrl C

### Question 5-2 Voulez-vous quitter ? → exercice d'assimilation

Faites un programme qui :

1. demande à l'utilisateur s'il veut quitter le programme ? Si oui, l'utilisateur doit rentrer la valeur 0, sinon, il tape sur n'importe quelle autre chiffre.
2. si l'utilisateur tape rentre la valeur 0, le programme s'arrête.
3. Sinon, le programme réitère en revenant à l'étape 1.

Programme attendu :

```
/* Programme pour tester la structure "while" :
   - boucle 10 fois en affichant une valeur i incrementee a chaque iteration
   - affiche la valeur de i apres la derniere boucle.
*/
#include <stdio.h>

int main () {
    int quitter = 0 ;
    char reponse ;
    int i = 0 ;
    while (quitter == 0) {
        printf ("Voulez-vous quitter le programme ? ") ;
    scanf ("%c", &reponse) ;
    if (reponse == 'o') {
        quitter = 1 ;
    }
    }
    return 0 ;
}
```

### Question 5-3 Devine un nombre 2 → exercice d'entrainement

Reprendre l'exercice "Devine un nombre" et modifiez la boucle for de sorte à ce que la boucle s'arrête soit lorsque les 10 tentatives ont été utilisées soit lorsque le nombre a été trouvé. Quelque soit le cas, le programme devra afficher suivant le

nombre de tentatives utilisées pour deviner le nombre :

- 1 tentative : accuser l'utilisateur de tricherie ou de voyance paranormale et lui conseiller de jouer au loto.
- entre 2 et 5 tentatives : féliciter chaudement l'utilisateur
- entre 6 et 9 tentatives : dire à l'utilisateur que c'est pas mal
- 10 tentatives : dire à l'utilisateur que c'était tout juste
- si l'utilisateur n'a pas trouvé, lui dire que c'est un gros nul ;-)

Programme attendu :

```
#include <stdio.h>

int main () {
    int secret = 42 ;
    int tentative = 0 ;
    int devine ;
        int trouve = 0 ;
    printf ("Le concepteur du programme a code un nombre secret entre 0 et 100\n") ;

    while ((trouve == 0) && (tentative < 10)) {
        printf ("Devinez le nombre (%d tentatives restantes)\n : ", tentative) ;
        scanf ("%d", &devine) ;

        if (devine < secret) {
            printf ("Le nombre secret est plus grand !\n") ;
        }
        else if (devine > secret) {
            printf ("Le nombre secret est plus petit !\n") ;
        }
        if (devine == secret) {
            printf ("Vous avez gagne !\n") ;
            trouve = 1 ;
        }
        tentative ++ ;
    }

    if (trouve == 1) {
        if (tentative == 1) {
            printf ("Vous Ãªtes soit un voyant soit un tricheur. Jouez au loto !\n") ;
        }
        else if (tentative < 5) {
            printf ("Toutes mes felicitations\n") ;
        }
        else if (tentative < 9) {
            printf ("Pas mal\n") ;
        }
        else if (tentative == 10) {
            printf ("Tout juste !\n") ;
        }
    }
    else {
        printf ("Gros nul ! Tu as perdu\n") ;
    }

    return 0 ;
}
```

### Question 5-4 Nombre premier → exercice d'entrainement

Déterminer si un nombre est premier ou non (rappel : un nombre premier n'est divisible que par 1 et par lui-même).
Programme attendu :

```c
#include <stdio.h>


/* Affiche les nombres premiers entre 1 et 100 */
int main () {
    int nb  ;
    int diviseur ;


    nb = 53 ;

    if ((nb == 1) || (nb == 2)) {
        printf ("%d est premier\n", nb) ;
        return 0 ;
    }

    diviseur = 2 ;

    do {
        diviseur ++ ;

    } while ((nb % diviseur) != 0 ) ;

    /* si diviseur est arrive jusqu'a nb sans jamais reussir a le diviser,
           alors le nombre est premier */
    if (diviseur == nb) {
        printf ("%d est premier\n", nb) ;
        return 0 ;
    }
    else {
        printf ("%d est n'est pas premier, il est divisible par %d\n", nb, diviseur) ;
        return 0 ;
    }
}
```

### Question 5-5 Nombre parfait

→ exercice d'entrainement

Déterminer si un nombre est parfait : un nombre est dit parfait lorsqu'il est égal à la somme de ses diviseurs (1 est considéré comme un diviseur mais pas le nombre lui-même). Exemple : 6 est parfait car 1, 2 et 3 sont ses diviseurs et que $1 + 2 + 3 = 6$. Programme attendu :

```c
#include <stdio.h>


/* Affiche les nombres premiers entre 1 et 100 */
int main () {
    int nb  ;
    int diviseur ;
    int somme ;


    nb = 496 ;

    if ((nb == 1)) {
        printf ("%d est parfait\n", nb) ;
        return 0 ;
    }

    diviseur = 1 ;
    somme = 0 ;

    while (diviseur < nb) {
        if ((nb % diviseur) == 0 ) {
            somme = somme + diviseur ;
        }
        diviseur ++ ;
    }

    if (somme == nb) {
        printf ("%d est parfait\n", nb) ;
        return 0 ;
    }
    else {
        printf ("%d est n'est pas parfait, la somme de ses diviseurs est %d\n", nb, somme) ;
        return 0 ;
    }
}
```

### Question 5-6 Nombres premiers

→ exercice d'entrainement

En réutilisant l'exercice 5-4, afficher les 100 premiers nombres premiers.

Programme attendu :

```
#include <stdio.h>


/* Affiche les nombres premiers entre 1 et 100 */
int main () {
    int nb  ;
    int diviseur ;
    int i ;

    for (i = 1 ; i <= 100 ; i ++) {
        nb = i ;

        if ((nb == 1) || (nb == 2)) {
            printf ("%d est premier\n", nb) ;
            continue ;
        }

        diviseur = 2 ;

        do {
            diviseur ++ ;

        } while ((nb % diviseur) != 0 ) ;

        /* si diviseur est arrive jusqu'a nb sans jamais reussir a le diviser,
           alors le nombre est premier */
        if (diviseur == nb) {
            printf ("%d est premier\n", nb) ;
        }
    }
}
```

### Question 5-7 Nombres parfaits → exercice d'entrainement

En réutilisant l'exercice 5-5, déterminer parmi les 10 000 premiers nombres entiers ceux qui sont parfaits [1] : afficher tous les nombres parfaits inférieurs à 10 000. Programme attendu :

1. les nombres parfaits à trouver sont 6, 28, 496 et 8128.

```c
#include <stdio.h>


/* Affiche les nombres premiers entre 1 et 100 */
int main () {
    int nb  ;
    int diviseur ;
    int somme ;
    int i ;

    for (i = 0 ; i <= 10000 ; i ++) {
        nb = i ;

        if ((nb == 1)) {
            printf ("%d est parfait\n", nb) ;
            continue ;
        }

        diviseur = 1 ;
        somme = 0 ;

        while (diviseur < nb) {
            if ((nb % diviseur) == 0 ) {
                somme = somme + diviseur ;
            }
            diviseur ++ ;
        }

        if (somme == nb) {
            printf ("%d est parfait\n", nb) ;
        }
    }
}
```

### Question 5-8 Suite de Syracuse

→ exercice d'entrainement

Soit la définition suivanteOn appelle suite de Syracuse une suite d'entiers naturels définie de la manière suivante : On part d'un nombre entier plus grand que zéro ; s'il est pair, on le divise par 2 ; s'il est impair, on le multiplie par 3 et on ajoute 1. [2] :

2. source : http ://fr.wikipedia.org/wiki/Conjecture_de_Syracuse

Affichage attendu :

```
u0 = 22
u1 = 11
u2 = 34
u3 = 17
u4 = 52
u5 = 26
u6 = 13
u7 = 40
u8 = 20
u9 = 10
u10 = 5
u11 = 16
u12 = 8
u13 = 4
u14 = 2
u15 = 1
```

Cette suite finit par atteindre la valeur 1 quel que soit le $U_0$ initial : afficher les termes de cette suite jusqu'à obtenir la valeur 1 pour $U_{n+1}$. Programme attendu :

```c
#include <stdio.h>

int main(void) {
    int N ;
    int un ;

    N = 7 ;
    un = N ;

    while (un != 1) {
        if ((un % 2) == 0) {
            /* si un est pair */
            un = un / 2 ;
        }
        else {
            un = 3 * un + 1 ;
        }
        printf ("un = %d \n", un);
    }
    return 0 ;
}
```

# 4 Validation des compétences acquises à l'issue de cette séance

Je maitrise les compétences demandées à l'issue de cette séance si **je suis capable** de :

- ☐ utiliser la boucle `while` pour exécuter une série d'instruction tant qu'une condition est vérifiée
- ☐ utiliser la boucle `do...while` pour exécuter une série d'instruction jusqu'à ce qu'une condition soit vérifiée
- ☐ expliquer la différence entre une boucle `while` et une boucle `do...while`