

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 opérateurs “&&” (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.
2. Que se passe-t-il si vous “oubliez” l'instruction `i = i + 1` ?
3. Initialisez `i` à 10 dans les deux programmes, que se passe-t-il ?
4. Pourquoi ?

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.

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

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 ;-)

Question 5-4 Nombre premier → exercice d'entraînement

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

Question 5-5 Nombre parfait → exercice d'entraînement

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 $1 + 2 + 3 = 6$.

Question 5-6 Nombres premiers → exercice d'entraînement

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

Question 5-7 Nombres parfaits → exercice d'entraînement

En réutilisant l'exercice 5-5, déterminer parmi les 10 000 premiers nombres entiers ceux qui sont parfaits¹ : afficher tous les nombres parfaits inférieurs à 10 000.

Question 5-8 Suite de Syracuse → exercice d'entraînement

Soit la définition suivante On 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

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} .

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

Je maîtrise 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`

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

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