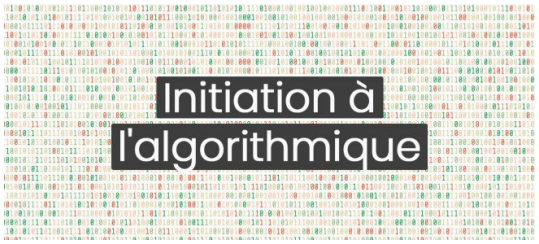
Chapitre 3 : Structures répétitives (boucles)

ALGORITHMIQUE (ALSD)



Préparé par Dr. BOUCHEBBAH Fatah

UNIVERSITÉ DE BÉJAIA

FACULTÉ DES SCIENCES EXACTES

DÉPARTEMENT D'INFORMATIQUE

Octobre 2022
Version 4

Table des matières

Objectifs

A l'issue de ce chapitre, vous serez capable de :

- Définir une instruction répétitives.
- Différencier entre les divers types de boucles.
- Utiliser pertinemment les instructions répétitives dans un algorithme.

Introduction

Imaginons que nous souhaitons calculer la moyenne d'un étudiant pour un module donné. Pour ce faire, nous écrivons un algorithme qui calcule cette moyenne en utilisant sa note obtenue à l'examen, ayant un coefficient de 2, et sa note de contrôle continu ayant un coefficient de 1. De ce faite, nous proposons l'algorithme suivant :

```
1 ALGORITHME Moyenne_étudiant ;
2 CONST
3 CoefExam = 2 ;
4 CoefEval = 1 ;
5 VAR NotExam, NotEval, Moy : Réel ;
6 DÉBUT
7  ÉCRIRE ('Donnez la note d'examen : ') ;
8  LIRE (NotExam) ;
9  ÉCRIRE ('Donnez la note d'évaluation : ') ;
10  LIRE (NotEval) ;
11  Moy <- (NotExam * CoefExam + NotEval * CoefEval)/ (CoefExam + CoefEval) ;
12  ÉCRIRE ('La moyenne = ', Moy ) ;
13 FIN.
```

Imaginons maintenant que nous voulons calculer la moyenne des étudiants de toute une section de 100 étudiants. Donc, le traitement illustré dans l'algorithme ci-dessus (Moyenne_étudiant) se répétera pour chaque étudiant de la section (*i.e.* 100 fois) d'une manière séquentielle. De plus, nous aurons besoin de 100 copies de chaque variable pour traiter les 100 étudiants de la section. Ce qui engendre un algorithme volumineux qui n'est pas agréable à voir, et difficile à comprendre.

Dans ce cas, nous pouvons utiliser une catégorie d'instructions de contrôle, appelée *structures répétitives (ou boucles)*.

II

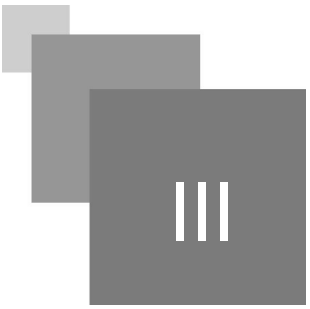
Définition d'une structure répétitive (boucle)

Une structure répétitive (aussi nommée une boucle) est une structure de contrôle qui répète l'exécution d'un même traitement, écrit une seule fois, autant de fois que c'est nécessairep.23 . Le nombre de répétitions à faire (aussi appelé tours de boucle) est déterminé soit par un nombre entier déclaré à l'avance, soit par une condition d'entrée (de sortie) à (de) la bouclep.23 .

En algorithmique (programmation), il existe trois types de structures répétitives : la boucle POUR ... FAIRE, la boucle TANT QUE... FAIRE, et la boucle RÉPÉTER ... JUSQU'À.

Il est à noté que le choix d'utilisation d'une telle ou telle boucle est déterminé en prenant en considération plusieurs facteurs, essentiellement la nature du problème à résoudrep.23 .

La boucle POUR ... FAIRE (FOR)



Dans cette section, nous nous intéressons au premier type de boucles, à savoir la boucle POUR ... FAIRE. Pour la présenter, nous donnons sa description ainsi que sa syntaxe algorithmique et en langage C.

1. Description de la boucle

Nous utilisons cette boucle lorsque *nous savons au préalable le nombre de tours de boucle à faire* p.23 . C'est le type de boucle que nous utilisons pour calculer la moyenne de 100 étudiants, ou pour éditer les fiches de paie de 1000 ouvriers par exemple.

Syntaxe : La boucle POUR ... FAIRE

La syntaxe de la boucle POUR ... FAIRE est la suivante :

```
1 POUR compteur ALLANT DE valeur_initiale A valeur_finale FAIRE
2  Traitement ;
3 FIN POUR ;
```

Ou en langage C :

```
1 FOR (initialisation_du_compteur; expression_de_test; mise_à_jour_du_compteur)
2 {
3      Traitement ;
4 }
```

Où :

- compteur : est une variable qui compte le nombre de tours de boucle effectués.
- valeur_initiale : est une valeur entière qui spécifie la valeur par laquelle on commence à compter le nombre de tours de boucle effectués.
- valeur_finale : est une valeur entière qui délimite le nombre de tours de boucle à faire.

Remarque : Plus de détails !

Au départ, la variable *compteur* prend la valeur de *valeur_initiale*. En suite, sa valeur s'incrémente automatiquement de 1 à chaque tour de boucle, jusqu'à ce qu'elle dépasse la valeur de *valeur_finale*. De ce fait, le traitement est répété pour chacune des valeurs de *compteur* comprises dans l'intervalle [

valeur_initiale, *valeur_finale*].

Exemple : Affichage de plusieurs valeurs

Afficher tous les entiers compris entre 50 et 100.

```
ALGORITHME Affiche ;
VAR i : Entier ;
DÉBUT
 POUR i ALLANT DE 50 A 100 FAIRE
  ÉCRIRE (i);
 FIN POUR ;
FIN.
```

Exemple : Amélioration de l'algorithme Moyenne_étudiant

Présenter une version de l'algorithme Moyenne_étudiant qui calcule la moyenne d'un module pour *n* étudiants. Les étudiants seront numérotés de 1 à n.

```
ALGORITHME Moyenne_n_étudiants ;
CONST
CoefExam = 2 ;
CoefEval = 1 ;
VAR
NotExam, NotEval, Moy : Réel ;
DÉBUT
 ÉCRIRE (‘Combien d'étudiants ? :') ;
 LIRE (n) ;
 POUR i ALLANT DE 1 A n FAIRE
  ÉCRIRE  (‘L'étudiant N° ‘, i) ;
  ÉCRIRE  (‘Donnez la note d'examen : ‘) ;
  LIRE (NotExam) ;
  ÉCRIRE  (‘Donnez le note d'évaluation : ‘) ;
  LIRE (NotEval) ;
  Moy <- (NotExam * CoefExam + NotEval * CoefEval) / (CoefExam +CoefEval) ;
  ÉCRIRE (‘La moyenne = ', Moy) ;
 FIN POUR ;
FIN.
```

Exercice

IV

[solution n°1 p.20]

On peut utiliser la boucle POUR ... FAIRE pour afficher la table de multiplication d'un entier introduit par l'utilisateur.

- Vrai.
- Faux.

Exercice

[solution n°2 p.20]

On peut calculer la somme d'une succession d'entiers introduits progressivement (un par un) à chaque tour de boucle.

- Vrai.
- Faux.

La boucle TANT QUE ... FAIRE (WHILE)

VI

Dans cette section, nous décrivons le deuxième type de boucle (*i.e.* la boucle TANT QUE ... FAIRE). De plus, nous exhibons sa syntaxe algorithmique et nous présentons sa structure en langage C.

1. Description de la boucle

La boucle TANT QUE .... FAIRE est basée sur *une condition d'entrée* à la boucle et non pas sur le nombre de répétitionsp.23 . Explicitement, *le traitement de la boucle est répété tant que la condition d'entrée est satisfaite* (retourne Vrai), et on s'arrête dès que la condition d'entée retourne Faux. La boucle TANT QUE .... FAIRE est plus générale que la boucle POUR ... FAIRE, et peut être utilisée dans plus de cas.

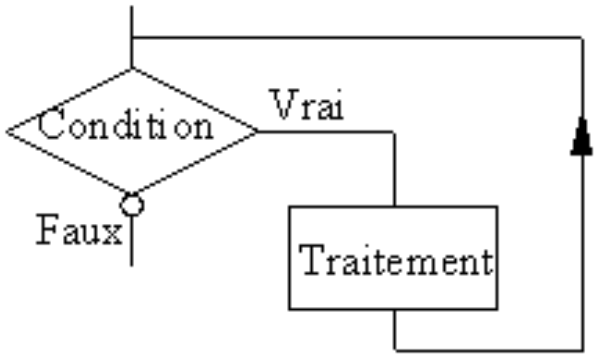
Syntaxe : La boucle TANT QUE .... FAIRE

La syntaxe de la boucle TANT QUE .... FAIRE est la suivante :

```
TANT QUE (condition) FAIRE
 Traitement ;
FIN TANT QUE ;
```

Ou en langage C :

```
WHILE (condition)
{
  Traitement ;
} ;
```



Organigramme de la boucle TANT QUE... FAIRE.

Remarque : Plus de détails !

- La boucle TANT QUE ... FAIRE comporte trois partie obligatoires, à savoir : *l'initialisation de la condition*, *vérification de la condition*, et *mise à jour de la condition*.
- La boucle POUR ... FAIRE peut être transformée en une boucle TANT QUE ... FAIRE. Mais, le *contraire n'est pas toujours réalisable*.
- Si on transforme une boucle POUR ... FAIRE en une boucle TANT QUE ... FAIRE, l'initialisation et l'incrémentation du compteur de répétitions doit se faire manuellement dans la nouvelle forme de la boucle. De plus, l'incrémentation peut se faire avec n'importe quel pas souhaité.

Exemple : Affichage de plusieurs valeurs (autre méthode)

Nous reprenons le même exemple ci-dessus, mais cette fois, la solution est donnée avec la boucle TANT QUE ... FAIRE.

```
1 ALGORITHME Affiche ;
2 VAR  n : Entier ;
3 DÉBUT
4  n <- 50 ;           (* Initialisation de la condition*)
5  TANT QUE (n <= 100) FAIRE    (* Vérification de la condition*)
6   ÉCRIRE (n) ;
7   n <- n+1 ;       (* Mise à jour de la condition*)
8  FIN TANT QUE ;
9 FIN.
```

Attention : Être vigilant !

Dans une boucle TANT QUE ... FAIRE :

- Si nous oublions la mise à jour de la condition, nous obtenons une boucle infinie (une boucle exécutée sans arrêt).
- Si condition d'entrée à la boucle est fausse dès le départ on n'entre pas dans la boucle et l'action ne sera jamais exécutée.

Exemple : Calcul de sommes

Calculer les sommes suivantes : S1 = 1+2+3+.....+n, et S2 =1^2+3^2+5^2 ... +n^2.

```
1 ALGORITHME Somme ;
2 VAR  n, i, j, S1, S2 : Entier ;
3 DÉBUT
4  ÉCRIRE ('Donnez la valeur de n');
5  LIRE (n) ;
6  S1 <- 0 ;
7  i <- 1 ;
8  TANT QUE (i <= n) FAIRE
9   S1 <- S1 + i;
10  i <- i + 1;
11 FIN TANT;
12 S2 <- 0;
13 j <- 1;
14 TANT QUE (j <= n) FAIRE
```

```
15   S2 <- S2 + carré (j);
16   j <- j + 2;
17  FIN TANT;
18 FIN.
```

Exercice

VII

[solution n°3 p.20]

La boucle TANT QUE ... FAIRE se base sur une condition d'entrée, et une mise à jour automatique de la condition. La boucle s'arrête lorsque la condition n'est plus vérifiée.

- Vrai.
- Faux.

La boucle RÉPÉTER... JUSQU'À (REPEAT ... UNTIL)

VIII

Enfin, dans cette section, nous présentons le dernier type de boucles (*i.e.* la boucle RÉPÉTER ... JUSQU'À). Sa description et sa syntaxe sont données dans ce qui suit.

1. Description de la boucle

La boucle RÉPÉTER ... JUSQU'À est utilisée quand le traitement à faire doit se répéter au moins une foisp.23 . Elle se base sur un principe inverse de celui de la boucle TANT QUE ... FAIREp.23 . En fait, dans le cadre de RÉPÉTER ... JUSQU'À, la condition est vérifiée à la sortie de la boucle et non pas à son entrée. En d'autres termes, on rentre dans la boucle quelque soit la condition et on répète le traitement à faire jusqu'à ce que la condition soit vérifiée (retourne Vrai).

La boucle RÉPÉTER ... JUSQU'À est utilisée essentiellement pour le contrôle des valeurs, saisies par clavier, avant de lancer un traitementp.23 .

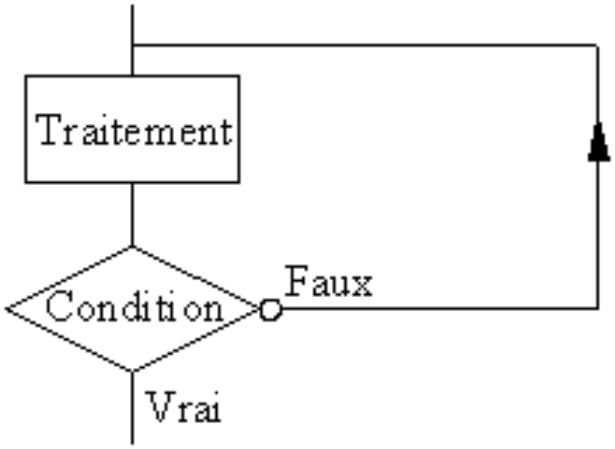
Syntaxe : La boucle RÉPÉTER ... JUSQU'À

La syntaxe de la boucle RÉPÉTER ... JUSQU'À est donnée comme suit :

```
1 RÉPÉTER
2  Traitement ;
3 JUSQU'À (condition) ;
```

Ou en langage C :

```
1 REPEAT
2  Traitement ;
3 WHILE (condition) ;
```



Organigramme de la boucle RÉPÉTER ... JUSQU'À.

Exemple : Nombre de divisions par 2

Écrire un algorithme qui permet à l'utilisateur de saisir un nombre pair et qui détermine combien de fois le nombre saisi est divisible par 2.

```
ALGORITHME  Pair-NbDIV2 ;
VAR N, N2, N3 : Entier ;
DÉBUT
 RÉPÉTER
  ÉCRIRE ('Donnez un entier pair') ;
  LIRE (N) ;
 JUSQU'À  (N MOD 2 = 0) ;
 N2 <- 0 ;
 N3 <- N ;
 RÉPÉTER
  N3 <- N3 DIV 2 ;
  N2 <- N2 +1 ;
 JUSQU'À  ((N3 MOD 2) <> 0) ;
 ÉCRIRE (N, ' est divisible par 2 ', N2, ' fois ') ;
FIN.
```

Complément : La boucle RÉPÉTER ... JUSQU'À pour le contrôle de saisie

Prenons le cas d'une saisie au clavier (une lecture) de notes qui sont comprises entre 0 et 20. Lors de la saisie, l'utilisateur, par erreur ou ignorance, risque de taper autre chose que la réponse attendue (*i.e.* une valeur comprise entre 0 et 20). Dès lors, le programme peut planter soit :

- *Par une erreur d'exécution :* parce que le type de réponse ne correspond pas au type de la variable attendu. Exemple : l'utilisateur tape une lettre ou un mot alors que le type de la variable note est de type Réel.
- *Par une erreur fonctionnelle :* il se déroule normalement jusqu'au bout, mais en produisant des résultats erronés. Exemple : l'utilisateur tape une note non comprise entre (0 et 20), il tape par exemple 23, -5, etc.

Alors, dans tout programme, on doit mettre en place ce qu'on appelle *un contrôle de saisie*, afin de vérifier que les données entrées au clavier correspondent bien à celles attendues par l'algorithme.

Le principe général du contrôle de saisie consiste à mettre la lecture de la variable à lire dans une boucle dont on ne sort pas jusqu'à ce que l'utilisateur tape une valeur qui correspond à ce que nous voulons. Nous pouvons utiliser soit la boucle TANT QUE ... FAIRE ou la boucle RÉPÉTER ... JUSQU'À pour faire un contrôle de saisie. Mais la boucle RÉPÉTER ... JUSQU'À est la mieux adaptée pour çà.

Exemple : Saisir une note d'un module

Faire un contrôle de saisie sur la lecture d'une note d'un examen.

```
1 RÉPÉTER
2  ÉCRIRE ('Tapez la note : ') ;
3  LIRE (note) ;
4 JUSQU'À ((note >= 0) ET (note ≤20));
```

Exercice

IX

[solution n°4 p.20]

Choisir la (les) bonne(s) réponse(s).

- ☐ La boucle RÉPÉTER ... JUSQU'À est basée sur le nombre de tours de boucle.
- ☐ La boucle RÉPÉTER ... JUSQU'À est basée sur une condition de sortie.
- ☐ La boucle RÉPÉTER ... JUSQU'À est similaire à la boucle POUR ... FAIRE.
- ☐

Contrairement à la boucle TANT QUE ... FAIRE, la boucle RÉPÉTER ... JUSQU'À doit boucler au moins une fois.

Synthèse et comparaison des différents types boucles

X

Pour finaliser, nous donnons dans le tableau ci-dessous une comparaison selon quelques critères entre les différentes structures répétitives que nous avions exposées dans ce chapitre.

Critère	POUR … FAIRE	TANT QUE … FAIRE	RÉPÉTER … JUSQU'À
Type de condition	Pas de condition	Condition d'entrée à la boucle	Condition de sortie de la boucle
Première entrée à la boucle	Si compteur ≤ Valeur_finale	Si condition = Vrai	Sans condition
Continuer à boucler	Tant que le compteur ≤ valeur_finale	Tant que condition = Vrai	Tant que condition = Faux
Sortie de la boucle	Quand compteur > valeur finale	Si condition = faux	Si condition = Vrai
Initialisation	Compteur = valeur_ initial	On doit le faire avant l'entrée	On doit le faire dans la boucle
Mise à jour de la condition	Automatique (pas = 1)	On doit le faire dans chaque itération	On doit le faire dans chaque itération
Utilisation	Nombre d'itérations connu	Quand on ne connait pas le nombre d'itérations	Quand on doit executer un traitement Au moins une fois.
Nombre de répétitions	Valeur_finale – valeur_initial +1	0 ou plusieurs fois	Au moins une fois

Comparaison entre les différents types de boucle.

Test

XI

Exercice

[solution n°5 p.21]

Tous les types de boucles sont des instructions de contrôle.

- O Vrai.
- O Faux.

Exercice

[solution n°6 p.21]

Que permet de faire une boucle ?

- ☐ Attribuer une valeur à une variable.
- ☐ Répéter certaines actions plusieurs fois.
- ☐ Initialiser les constantes d'un algorithme.
- ☐ Améliorer la vitesse d'exécution d'un programme mais en augmentant le risque d'erreurs.
- ☐

Réduit la longueur d'un algorithme en évitant la réécriture d'une même (plusieurs) instruction(s) plusieurs fois.

Exercice

[solution n°7 p.21]

Peut-on insérer une boucle dans une autre ?

- O Oui.
- O Non.

Solutions des exercices

> **Solution** n°1

Exercice p. 8

On peut utiliser la boucle POUR ... FAIRE pour afficher la table de multiplication d'un entier introduit par l'utilisateur.

- ⊙ Vrai.
- ○ Faux.

Dans ce cas, on peut utiliser la boucle POUR ... FAIRE. Car, le nombre de tours de boucle à faire est connu.

> **Solution** n°2

Exercice p. 9

On peut calculer la somme d'une succession d'entiers introduits progressivement (un par un) à chaque tour de boucle.

- ○ Vrai.
- ⊙ Faux.

Dans ce cas, on ne peut pas utiliser la boucle POUR ... FAIRE. Car, le nombre de tours de boucle à faire n'est connu à l'avance. Dans ce cas, on doit utiliser un autre type de boucle.

> **Solution** n°3

Exercice p. 13

La boucle TANT QUE ... FAIRE se base sur une condition d'entrée, et une mise à jour automatique de la condition. La boucle s'arrête lorsque la condition n'est plus vérifiée.

- ○ Vrai.
- ⊙ Faux.

La condition doit être mise à jour par une instruction particulière déclarée à l'intérieure de la boucle.

> **Solution** n°4

Exercice p. 17

Choisir la (les) bonne(s) réponse(s).

- ☐ La boucle RÉPÉTER ... JUSQU'À est basée sur le nombre de tours de boucle.
- ☑ La boucle RÉPÉTER ... JUSQU'À est basée sur une condition de sortie.
- ☐ La boucle RÉPÉTER ... JUSQU'À est similaire à la boucle POUR ... FAIRE.

☑

Contrairement à la boucle TANT QUE ... FAIRE, la boucle RÉPÉTER ... JUSQU'À doit boucler au moins une fois.

> **Solution** n°5

Exercice p. 19

Tous les types de boucles sont des instructions de contrôle.

- ◉ Vrai.
- ○ Faux.

> **Solution** n°6

Exercice p. 19

Que permet de faire une boucle ?

- ☐ Attribuer une valeur à une variable.
- ☑ Répéter certaines actions plusieurs fois.
- ☐ Initialiser les constantes d'un algorithme.
- ☐ Améliorer la vitesse d'exécution d'un programme mais en augmentant le risque d'erreurs.

☑

Réduit la longueur d'un algorithme en évitant la réécriture d'une même (plusieurs) instruction(s) plusieurs fois.

> **Solution** n°7

Exercice p. 19

Peut-on insérer une boucle dans une autre ?

- ◉ Oui.

- Non.

Références

1

L. Baba-Hamed, S. Hocine. Algorithmique et structures de données : Cours et exercices avec solutions. Office des Publications Universitaires, 2006

2

M. C. Belaid. Algorithmique et programmation en Pascal, cours, exercices et travaux pratique avec corrigés. Edition Les Pages Bleues, 2004.

3

M. C. Belaid. Algorithme et programmation en Pascal. Edition les pages bleus, 2006.

4

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Algorithmique : Cours avec 957 exercices et 158 problèmes. Édition DUNOD, 3ème édition, 2010.

5

J. Courtin. Initiation à l'algorithmique et aux structures de données. Edition DUNOD, 1998.

6

M. Divay. Algorithmes et structures de données génériques. Edition Dunod, 2004.

7

L. Goldschlager and A. Lister. Informatique et algorithmique. InterEditions. 1986.

8

https://9alami.info/cours-informatique/liste-des-modules/lecon1-notion-dalgorithme (Consulté le 09/03 /2021)

9

https://www.youtube.com/playlist?list=PL2aehqZh72Lumvy4tSekr6Rzcgwn15MLI (Consulté le 05/04 /2021)