

PSEUDOCODE ET LANGAGE JAVA :

LES STRUCTURES DE CONTRÔLES :

LA RÉPÉTITION

INTRODUCTION

Nous avons vu dans le cours précédent l'utilisation des structures conditionnelles. Celles-ci ne suffisent pas dans bien des cas pour réaliser nos algorithmes.

Prenons un exemple : Prenons le cas d'une saisie au clavier (une lecture), où par exemple, le programme pose une question à laquelle l'utilisateur doit répondre par O (Oui) ou N (Non). Il faut, évidemment, mettre en place un contrôle de saisie, afin de vérifier que les données entrées au clavier correspondent bien à celles attendues par l'algorithme.

```
Var Rep : Caractère
Début
    Afficher "Voulez-vous un café ? (O/N)"
    Saisir Rep
    Si Rep <> "O" et Rep <> "N" Alors
        Afficher "Saisie erronée. Recommencez"
        Saisir Rep
    FinSi
Fin
```

C'est impeccable. Du moins tant que l'utilisateur a le bon goût de ne se tromper qu'une seule fois, et d'entrer une valeur correcte à la deuxième demande ! La solution consistant à aligner des Si... en pagaille est donc une impasse.

La bonne solution consiste ici à reposer la question **tant qu'**il n'apporte pas une réponse correcte.

Ces structures se montrent indispensables dans de très nombreux cas.

En voici d'autres exemples :

- Chercher dans un annuaire le numéro de téléphone d'une personne dont on connaît le nom. Le nombre de répétitions dépend de la longueur de la liste, mais également de la position du nom dans la liste.
- Calculer la N^{ème} puissance d'un nombre "a" par multiplication successive du nombre a par lui-même.

Il existe 3 formes de répétitions en algorithmique, même chose en JAVA, qui s'utilisent selon des critères bien précis :

En ALGO	En JAVA	Critère d'utilisation
Le POUR :	for	S'utilise lorsque le nombre d'itérations (n) est connu à l'avance, donc <u>fixe</u>
Le TANTQUE :	while	S'utilise lorsque le nombre d'itérations (n) est inconnu (<u>de 0 à +∞</u>)
Le RÉPÉTER JUSQU'A:	do ... while	S'utilise lorsque le nombre d'itérations (n) est inconnu, mais ce fait au moins 1 fois (<u>de 1 à +∞</u>)

Seule la boucle "TANT QUE" est fondamentale. Avec cette boucle, on peut réaliser toutes les autres boucles.

RÉPÉTITIONS AVEC TANTQUE ... FAIRE.

La boucle TANT QUE ... FAIRE permet de répéter un traitement tant qu'une **expression conditionnelle** est vraie. Si lors du tout premier test, la condition n'est pas vraie, le traitement ne sera pas exécuté. Le nombre d'itérations est donc compris entre 0 .. +∞
Donc attention, si la condition est toujours VRAIE on fait une **boucle infinie** !

Pseudolangage:	Comportement :	EN JAVA:
<pre>TANTQUE condition évaluée à vrai FAIRE action1 action2 FINTANTQUE</pre> <p style="text-align: center;">} Séquence A</p>		<pre>while (condition) { instruction1; instruction2; }</pre>

Que ce soit en pseudocode ou en JAVA, la problématique principale est la même : il faut trouver la condition. La condition que l'on va mettre dans le TANTQUE exprime les raisons de continuer à boucler. Il faut donc se poser cette question :

1. **À quelle condition continue-t-on de boucler ?** Réponse à la question : **je continue tant que ...**

L'expérience montre que pour certaines personnes il est parfois plus facile de trouver la raison pour arrêter la boucle. Dans ce cas la question à se poser est :

2. **À quelle condition arrête-t-on de boucler ?** Réponse à la question : **je m'arrête quand la condition est...**

Prenons un exemple pour illustrer ses 2 questions : on veut écrire un algorithme permettant d'afficher les 11 premiers nombres (de 0 à 10). Pour résoudre ce problème, on utilisera une variable "i"(compteur) comme variable de boucle.

Au départ (i) vaut 0

- La réponse à la question 1 est : **je continue tant que** $i \leq 10$
- La réponse à la question 2 est : **je m'arrête quand** $i > 10$

Ce que l'on mettra comme condition à notre boucle c'est la réponse à la question 1. Mais si on est plus à l'aise avec la question 2, il suffira d'inverser le résultat et notre condition deviendra : $\text{NON}(i > 10)$ ce qui revient à écrire $i \leq 10$

Ce qui donne:

<pre> VAR i : entier DÉBUT i ← 0 TANTQUE i ≤ 10 FAIRE Afficher i i ← i + 1 FIN TQ FIN </pre>	<pre> public static void boucleWhile() { int i=0; while(i ≤ 10) { System.out.println(" " + i); i++; } } </pre>
--	--

Remarque : ici le nombre d'itérations est connu : le POUR est le plus adapté

EXERCICE AVEC LA BOUCLE WHILE

Exercice 1 : Écrire un algorithme qui calcule le cube des nombres que l'utilisateur saisit. Pour arrêter, l'utilisateur doit entrer la valeur 0. Si l'utilisateur saisit 0 le programme s'arrête. Le cube de 0 n'est pas affiché. Dans cet exercice le nombre de répétitions n'est pas connu à l'avance. L'utilisateur a tout à fait le droit de saisir 0 dès la 1^{re} saisie. Il est donc logique que problème soit géré avec un TANTQUE.

Exercice 2 : écrire un programme qui calcule la somme des entiers de 1 à 100.

Pour réaliser ce problème, on a tendance à réfléchir comme on le ferait en math. Le résultat c'est $1+2+3+4+5+6+...+99+100$.

Dans un programme on ne pourra faire comme cela. En réalité cet exercice correspond à un problème dit de CUMUL. Plutôt que de calculer en ligne, on procèdera par résultat intermédiaire. On calcul $1+2 \rightarrow 3$, puis $3+3 \rightarrow 6$, puis $6+4 \rightarrow 10$, puis $10+5 \rightarrow 15...$

Exercice 3 : reprendre l'exercice précédent, mais cette fois-ci l'utilisateur saisit la valeur limite (N). Donc on calculera la somme des entiers de 1 à N. Cette notation avec la lettre "N" est très souvent utilisé dans les algorithmes, pour indiquer que l'on fera N fois la boucle. N est ici une inconnue au moment où l'on écrit notre programme. Sa valeur ne sera connue qu'au moment de l'exécution du programme. Ici N dépend de la saisie de l'utilisateur.

Exercice 4 : écrire un programme qui permet à un utilisateur de saisir un code PIN (code à 4 chiffres). L'utilisateur a le droit à 5 essais maximum pour saisir son code. Au-delà un message de refus, et si le code est correct un message de bienvenue.

RÉPÉTITION AVEC RÉPÉTÉE...TANTQUE

Le **RÉPÉTER ...TANTQUE** permet de répéter un traitement tant qu'une expression conditionnelle soit évaluée à vrai. Le test se fait en fin de boucle, donc la séquence A est exécutée au moins une fois. Le nombre d'itérations est donc compris entre 1 ... $+\infty$. Parce qu'elles sont exécutées au moins une fois, les boucles RÉPÉTÉES ... TANTQUE est bien adapté aux contrôles de saisie. Attention comme avec une boucle TANTQUE, si la condition est toujours VRAI → boucle infinie.

Pseudo langage:	Comportement :	EN JAVA:
RÉPÉTER <div style="display: flex; align-items: center;"> <div style="border-left: 1px solid black; padding-left: 10px; margin-right: 10px;"> action1 action2 </div> Séquence A </div> TANTQUE condition évaluée à vrai		<pre>do { instruction1; instruction2; }while (condition) ;</pre>

Au niveau algorithmique le REPETER ... TANTQUE n'existe pas. On utilise une syntaxe qui est le REPETER ... JUSQU'A. Pour simplifier les choses, nous n'utiliserons pas cette dernière syntaxe.

Au niveau du langage JAVA, le mot-clé utiliser et do {...} while(condition); on note les similitudes avec le while(condition) {...}, à ceci près la présence d'un ; à la fin du **while**. Celui-ci est obligatoire, et c'est la seule structure de contrôle qui a besoin de ce point-virgule de fin.

Attention 1 : ne pas mettre un ; à la fin d'une boucle TANTQUE, SI, SUIVANT, et POUR. Dans le programme ci-contre un ; a été ajouté à la fin du while. Si aucune erreur n'est générée au niveau de la syntaxe et ne semble donc tout à fait cohérente, ce programme boucle à l'infini.

```
int i = 0;
while (i < 10);
{
    System.out.println(i);
    i = i + 1;
}
```

Explication dans le code en commentaire :

```
int i = 0;
while (i <= 10)
    ; // ici j'exécute une instruction vide. Comme i n'est pas modifié, la condition sera
    // toujours évaluée à VRAI

{
    // le bloc d'instruction est indépendant, et n'est pas rattaché au while. Ce bloc ne
    // sera jamais exécuté
    System.out.println(" " + i);
    i++;
}
```

Attention 2 : Si au moins 2 instructions dans la boucle ajouter les { } délimitant le bloc , sinon risque de boucle ∞ comme dans l'exemple

```
int i = 0;
while (i < 10)
    System.out.println(" " + i); // En l'absence des { } seulement la 1re ligne est
                                // rattaché au while.

// comme i n'est pas rattaché au while, i ne sera jamais modifié, la condition sera toujours
// évaluée à VRAI
i = i + 1; // l'instruction i=i+1 est en dehors du while, donc jamais exécuté.
```

Comme le montrent ces 2 exemples, on peut facilement faire une boucle infinie. Pour éviter ses pièges on retient :

- **Pas de ; à la fin d'un while(), mais un ; à la fin d'un do{}while();**
- **Programmer simplement en n'oubliant pas les { } du bloc while, do while, for**

EXERCICE AVEC DO{ .. }WHILE() ;

Exercice : reprendre l'exemple d'introduction (réponse O(Oui), N(Non)). Le gérer avec un RÉPÉTÉ ... TANTQUE.

Exercice : écrire un programme qui simule une caisse enregistreuse. La caissière rentre le prix d'un article, sa quantité, et ceux pour tous les articles. Le prix total est affiché lorsqu'il n'y a plus d'article. Pour cela le programme pose la question "Encore des articles ? O=Oui, N=Non"

RÉPÉTITION AVEC LE POUR

dans l'exercice précédent nous voyons bien que le nombre de produits n'est pas connu à l'avance (au moins un). Dans ces cas-là la boucle TANTQUE ou REPETER est bien adaptée. Lorsque le nombre de répétitions est connu à l'avance on dispose de la boucle POUR.

Le principe de cette boucle est d'utiliser une variable de boucle qui est affectée à une valeur **début**. Par convention on utilise les lettres (i), (j) et (k) pour ces variables de boucle qui sont des compteurs. À chaque tour dans la boucle on incrémente la variable de boucle d'un **pas** définie dans le POUR. Ce pas est en général de +1. On compte donc de 1 en 1, jusqu'à ce que (i) atteigne la valeur de **fin incluse**. Le pas peut être de 2,3,... mais aussi -1,-2 ,...

Pseudo langage:	Comportement :	EN JAVA:
Pour i ← début à fin pas incrément faire action1 action2 Fin Pour	Séquence A	<pre>for(i=début ; i<= fin ; i=i+1) { 1 2 3 inst1; inst2; }</pre>

Dans la syntaxe du JAVA on distingue 3 parties distinctes séparées par des ;

- **1** : initialisation de la variable (i). Ne se fait qu'une fois, avant la boucle.
- **2** : test d'arrêt de la boucle. Se fait à chaque tour de la boucle. Si la condition est VRAI on continue la boucle.
- **3** : incrémentation de la variable(i). Se fait à chaque tour de la boucle

Exemple : afficher les 10 premiers nombres pairs → 0,2,4,6,8,10,12,14,16,18

Pseudo langage:	EN JAVA :
Pour i ← 0 à 18 pas +2 faire Afficher(i) Fin Pour	<pre>int i; for(i=0 ; i<= 18 ; i=i+2) { System.out.println(" " + i); }</pre>

On pourrait faire la même chose avec une boucle TANTQUE, mais il faudrait initialiser la variable (i) et l'incrémenter explicitement. Le POUR fait la même chose, mais en plus simple.

```
i ← 0
TANT QUE i <= 18 FAIRE
    Afficher( i )
    i ← i + 2    // dans la boucle POUR cette ligne n'apparaît pas. L'incrément est automatique.
FINTANTQUE
```

EXERCICE BOUCLE FOR

Exercice 1 : écrire un programme qui calcule la somme des entiers de 1 à N avec une boucle FOR.

Exercice 2 : écrire un programme qui affiche la liste des nombres divisibles par (x) pour les N premiers nombres. Par exemple si N= 100 et que x vaut 6 on affichera la liste suivante de nombres :

0	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90
---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Exercice 3 : écrire un programme permettant de saisir N notes. À la fin du programme le programme affiche la moyenne des N notes.

L'INSTRUCTION BREAK

Comme avec le switch, on peut utiliser l'instruction break pour sortir d'une boucle while, for et do while. Cette instruction permet de quitter à tout moment une boucle. Notons cependant que dans un while() et un do{} while() on peut en général s'en passer.

Exemple :

```
String c;
int x=0;
while( x < 10 ){
    System.out.println(" " + x)
    System.out.println("Saisir c");
    c=sc.next();
    if(c=="q"){
        break; //pour sortir
    }
    x++;
}
```

FAIRE UNE TRACE D'EXÉCUTION

Il est souvent bien utile de réaliser ce que l'on appelle une trace d'exécution. Cela consiste à dérouler le programme à la main, en vérifiant son bon fonctionnement, ou bien pour comprendre ce qu'il fait.

Exemple: que fait ce programme ? Pour répondre, nous allons faire une trace d'exécution en complétant le tableau ci-dessous.

```
PROGRAMME Pgr1
VAR
    a, resultat: réel
    n, i : entier
DÉBUT
    resultat ← 1
    Lire a, n
    POUR i ← 1 à n FAIRE
        resultat ← resultat * a
    FINPOUR
    ECRIRE "Le resultat : ", resultat
FIN
```

Variable Opérations	a	n	i	i <= n	resultat
Initialisation des variables	0	0	0		0
resultat ← 1	0	0	0		1
Lire a, n	10	3	0		1
Pour	10	3	1	True	1
resultat ← resultat * a	10	3	1	True	10
resultat ← resultat * a			2	True	100
			3	True	1000
	10	3	4	False	1000
Écrire					

CONCLUSION

Grâce aux différentes structures de contrôles, on peut résoudre une grande partie des problèmes informatiques. Le problème est de faire le bon choix parmi toutes ces structures. Pour cela on peut suivre l'algorithme suivant :

```
Si le nombre de répétition est connu alors
    Choisir un "POUR"
Sinon si le nombre de répétition >= 1 alors
    Choisir un "REPETER"
Sinon
    Choisir un "TANTQUE"
fin
```