Chapitre 5

Instructions répétitives

1. Instruction TANT QUE

Syntaxe :

```
Tantque <condition> faire
    < séquence d'instructions >
finTantque
```

Mécanisme
la première fois :

évaluation de <condition>

- vrai ➔ la séquence d'instructions est exécutée
- faux ➔ on n'entre pas dans la structure et c'est l'instruction qui suit *Fin tant que* qui est exécutée

les fois suivantes, on vient d'exécuter la séquence d'instructions. Il y a retour sur la condition pour la réévaluer

- vrai ➔ la séquence d'instructions est exécutée
- faux ➔ on n'entre pas dans la structure et c'est l'instruction qui suit *Fin tant que* qui est exécutée

Exemple : voir algorithme intérêts Chapître 1 page 4

Remarques :

- Il faut que la condition soit **évaluable** au moment où elle est évaluée
- La séquence peut ne jamais être exécutée si la condition est fausse
- Il faut que la condition soit **modifiée** par la séquence d'instructions, sinon soit on n'entre jamais, soit la boucle est éternelle

exemple :

```
Début
1 a ← 1
2 b ← a
3 tantque a < 100 faire
4    b ← b * 5
5    écrire(b)
6 fintantque
Fin
```

	a	b	Ecran
1	1		
2		1	
3			
4		5	
5			5
3			
4		25	
5			25…

a < 100 toujours vrai donc boucle éternelle

Résumé

Instructions simples : agissent directement sur les variables.

	affectation	modifient certaines variables
interactivité avec utilisateur	**écriture**	
	lecture	

Instructions structurées : contrôlent l'exécution des instructions simples en fonction de l'état des variables à chaque instant du déroulement de l'algorithme.

Si <condition> **alors** <séquence_1> **sinon** <séquence_2> **finsi**	instruction conditionnelle
Tantque <condition> **faire** <séquence> **fintantque**	instruction répétitive

D'autres instructions structurées seront présentées dans la suite.

2. Instruction Répéter

L'instruction *Tantque* était contrôlée par une expression booléenne d'entrée.
On dispose d'une autre répétitive contrôlée par une expression booléenne de sortie

`tantque somme < sommeFinale faire`
`  < séquence >`
`fintantque`
	`répéter`
	`  < séquence >`
	`jusqu'à somme >= sommeFinale`

Syntaxe :

répéter
< séquence >
jusqu'à <expression booléenne>

Mécanisme :

- éxécution de la séquence une première fois
- évaluation de <expression booléenne>
  - *vrai* ➔ sortie et exécution de la suite de l'algorithme
  - *faux* ➔ retour pour exécuter à nouveau la séquence et tester l'expression booléenne.

exemple :

```
Algorithme Exemple

/* Lexique */
Variables
   nbCoups, d1, d2 : entier
   …

Début
   …
   nbCoups ← 0
   répéter
      lire(d1)
      lire(d2)
      nbCoups ← nbCoups + 1
   jusqu'à d1 + d2 = 12
   …
Fin
```

Remarques :

- la séquence doit modifier l'expression booléenne ou condition de sortie, sinon la boucle est éternelle
- la séquence est exécutée au moins une fois

3. Instruction Pour

Ici la répétitive est contrôlée par le comptage du nombre de répétitions de la séquence.

Syntaxe :

❶
pour <variable> de <expr_début> à <expr_fin> faire
    <séquence>
    ❷
finpour

Mécanisme :

pour i **de** debut **à** fin **faire** <séquence> **finpour**	≡	i ← debut **tantque** i <= fin **faire** <séquence> i ← i + 1 **fintantque**

au repère ❶

- la variable de boucle prend la valeur de <expr_début>
- l'expression booléenne **<variable> ≤ <expr_fin>** est évaluée
  - *vrai* ➔ la séquence est exécutée
  - *faux* ➔ exécution de l'instruction qui suit le fin pour

au repère ❷

- la variable est incrémentée
- il y a retour pour évaluer l'expression booléenne **<variable> ≤ <expr_fin>**
  - *vrai* ➔ la séquence est exécutée
  - *faux* ➔ exécution de l'instruction qui suit le fin pour

Simulation d'un exemple

```
1    n ← 0
2    s ← 0
3    pour i de 1 à 3 faire
4       n ← n + 5
5       s ← s + n
6    finpour
7    écrire (s)
```

	n	n	i	Expr. bool.	Ecran
1	0				
2		0			
3			1	1 ≤ 3 : *vrai*	
4	5				
5		5			
6			2		
3				2 ≤ 3 : *vrai*	
4	10				
5		15			
6			3		
3				3 ≤ 3 : *vrai*	
4	15				
5		30			
6			4		
3				4 ≤ 3 : faux	
7					30

Remarques :

- La séquence ne doit modifier :
  - ni la variable de boucle
  - ni l'expression de fin
- la séquence peut ne jamais être exécutée si, avant d'aborder l'instruction pour, <expr_début> est déjà strictement supérieure à <expr_fin>
- Il existe une instruction pour par pas décroissant :

Simulation d'un exemple

```
1    n ← 0
2    s ← 0
3    pour i de 1 à 3 faire
4       n ← n + 5
5       s ← s + n
6    finpour
7    écrire (s)
```

	n	n	i	Expr. bool.	Ecran
1	0				
2		0			
3			1	1 ≤ 3 : *vrai*	
4	5				
5		5			
6			2		
3				2 ≤ 3 : *vrai*	
4	10				
5		15			
6			3		
3				3 ≤ 3 : *vrai*	
4	15				
5		30			
6			4		
3				4 ≤ 3 : faux	
7					30

Remarques :

- La séquence ne doit modifier :
  - ni la variable de boucle
  - ni l'expression de fin
- la séquence peut ne jamais être exécutée si, avant d'aborder l'instruction pour, <expr_début> est déjà strictement supérieure à <expr_fin>
- Il existe une instruction pour par pas décroissant :

pour i **de** debut **à** fin **par pas de –1 faire** <séquence> **fi pour**	≡	i ← debut **tant que** i >= fin **faire** <séquence> i ← i - 1 **fintantque**

4. Récapitulatif

- Instructions simples
  - affectation
  - lecture
  - écriture

- Instructions structurées

  - instructions de choix
    - si alors sinon finsi
    - selon finselon
    - si alors sinonsi alors sinon finsi

  - instructions itératives
    - tant que - fin tant que : contrôle à l'entrée
    - répéter - jusqu'à : contrôle à la sortie
    - pour - fin pour : contrôle sur le nombre de répétitions

Remarque 1 :

Comment choisir la structure itérative adaptée ?

Nombre d'itérations connu avant l'exécution de la 1ère itération et non modifiable.	Nombre d'itérations non connu avant l'exécution de la 1ère itération.	
	0 itération possible	au moins 1 itération
Pour	**tantque**	**répéter**

Remarque 2 :

Voici 2 types d'algorithmes itératifs qui traitent des listes et qui se distinguent suivant que l'on doive traiter le dernier élément de la liste ou non. Les exemples choisis pour illustrer ces algorithmes abordent des circonstances très concrètes.

Exemple 1 :

A la caisse d'un supermarché, la caissière doit traiter tous les clients dans la file d'attente.

Exemple 2 :

A la caisse d'un supermarché, la caissière doit traiter (comptabiliser) tous les articles d'un client qui se trouvent sur le tapis roulant, sauf le dernier élément qui se trouve être la barre de séparation avec le client suivant.

Pour peu que l'on dispose de fonctions permettant d'obtenir l'élément suivant et de tester si cet élément est le dernier de la liste, voici le principe de ces 2 algorithmes.

```
/* exemple 1 */
/* tous les éléments sont traités */

répéter
   element = elementSuivant()
   traiter(element)
jusqu'à dernier(element)
```

```
/* exemple 2 */
/* tous les éléments sont traités sauf le dernier */

element = elementSuivant()
tantque Non dernier(element)
   traiter(element)
   element = elementSuivant()
finTantque
```